Interface

2003 August

[表紙デザイン：㈱プランニング・ロケッツ]

Contents 2003 August

特 集

動きのはやい技術動向をより効率的に理解するための用語解説

Basics of current computer technology

話題のテクノロジ解説

ショウレポート&コラム

一般解説＆連載

情報のページ

連載「XScaleプロセッサ徹底活用研究」，「家電機器をネットワーク化するアーキテクチャ Universal Plug and Playの全貌」，「開発環境探訪」は，お休みさせていただきます．

Contents 2003 August

日本最大級の Linux イベント

LinuxWorld Expo /Tokyo 2003

北村俊之

「オープンマインドが出会う場——身近な Linux」をテーマに，「LinuxWorld Expo/Tokyo 2003」が 5 月 21 日 (水) ～ 23 日 (金) の 3 日間，東京ビッグサイトで開催された．主催は (株) IDG ジャパン．

今年で第 5 回を迎える本展示会は，出展社数も 40 社以上で，昨年を上回る規模となった．昨年までは，Linux に対応したハードやソフトを紹介するという，製品よりの展示が中心だったが，今回は Linux ベースのサービスやソリューションに焦点を絞った展示が増えていた．当然ながら来場者の関心も「Linux で何ができるのか，どこまでできるのか」へと変化している．現在のシステムを Linux に乗せ換えたときのメリットは何か，コスト的には見合うのだろうか，安全かつ安定して稼動するのだろうか，と来場者の関心は高い．最終的な来場者数は 42,931 人だった．

● 各出展社のソリューション

日本ヒューレット・パッカードは，「HP serviceguard for Linux」による HA クラスタシステムや Itanium2 サーバによる「Oracle 9iRAC システム」，各種ブレードサーバなどエンタープライズ Linux を実現するためのソリューションを幅広く展示していた (**写真 1**)．また，同社のシステムを利用したパートナ各社のデモも数多く展示されており，来場者の関心も高いようだった．

〔写真 1〕 **Oracle9i Database for Linux/Itanium**

日本IBMは，xSeries (**写真 2**) をはじめとするエンタープライズサーバやブレードサーバの展示とデモを行っていた．また，先ごろ買収したラショナル製品を利用した，音声，画像配信の事例なども紹介されていた．

〔写真 2〕 **IBM の xSeries 335**

ゲメックスは，Linux サーバとしては世界最小クラスのフルスペックサーバ「Server The BOX」のニューモデル「B シリーズ」の展示を行っていた (**写真 3**)．本製品は専用のマザーボードを備え，振動によるプリント基板のたわみを排除した堅牢で耐久性の強い設計になっているという．また，ハードウェア RAID コントローラや LAN インターフェース，IrDA，PCI バス，パラレルなどの内部拡張インターフェースをそれぞれ装備している．標準の CPU は Celeron566MHz を採用しているが，オプションで Pentium III 700MHz の搭載も可能だという．

〔写真 3〕 **ゲメックスの Server The BOX**

大塚商会は，サイボウズのグループウェア「ガルーン」とクラスタソフトの「LifeKeeper」および「CyberFinder2」の連携ソリューションを中心とした展示およびデモを行っていた．NEC は，最新のテクノロジを活用した企業向けソリューションとして，「CLUSTERPRO for Linux」，「TX7/i6010」，「Express5800/BladeServer」を中心に展示を行っていた．「Express5800/BladeServer」は，サーバとしての機能を 1 枚で実現するブレードサーバとして，来場者の注目も高いとのことであった．

トレンドマイクロは，「ServerProtect for Linux」，「InterScan VirusWall アプライアンス」，「InterScan WebManager」といった Linux 用ウイルス対策ソフトの展示とデモを行っていた (**写真 4**)．Windows ほど露出度は高くないが，Linux をターゲットとしたウイルスは着実に増加しており，その手口も複雑化しているという．「ServerProtect for Linux」は，「Red Hat Linux 7.2/7.3」に対応したファイルサーバ専用のウイルス対策ソフトで，カーネルレベルでのウイルス検出が可能なほか，ウイルス感染の通知機能も装備している．

〔写真 4〕 **トレンドマイクロのブース**

沖データは，UNIX/Linux 環境のプリンタとして Windows や Macintosh 環境と同等の使い勝手を提供する「MICROLINE UNIX Printing System」の展示を行っていた．ポストスクリプト 3 互換インタプリタ搭載のマルチ OS に対応した A4 カラーページプリンタである「MICROLINE 5300」(**写真 5**) は人気の高い製品だという．

〔写真 5〕 **沖データの MICROLINE 5300**

東芝は，オールインワンアプライアンスサーバ「MAGNIA シリーズ」を中心に各種ソリューションの展示を行い，Linux 環境での帳票処理を円滑に行う「FlyingServ Web 帳票」や J2EE でインタラクティブな画面を実現する機能の紹介デモも行っていた．

日本オラクルとミラクル・リナックスの共同出展ブース (**写真 6**) では，オラクルが提唱する新しい Linux の可能性「Unbreakable Linux」をテーマに，製品，テクノロジ，導入コスト，TCO 削減，導入事例などを切り口に最新のソリューションを紹介するセミナが開催されていた．また，同社から提供されている「Oracle9i RAC」は，可用性や管理性，運用性に優れた共有ディスク方式を採用し，各ノード間でのキャッシュの同期化を実現するキャッシュフュージョン技術を実装したクラスタリングソフトウェアで，Oracle データベース上で開発した既存のアプリケーションを，ほとんど手を加えることなく，クラスタ対応にできるという．

〔写真 6〕 **日本オラクルとミラクル・リナックスの共同出展ブース**

ターボリナックスと SRA の共同ブースでは，クラスタや大規模データベースシステムを想定した Linux ソリューションのデモや事例紹介が行われていた．ソリューションゾーンでは，PostgreSQL をベースとしたオープンソースデータベースの展示が行われており，来場者の関心を集めていた．

バックボーン・ソフトウェアは，エンタープライズストレージ環境で，データのバックアップ/リストアを効率的に行うソフトウェアの最新バージョン「NetVault 7」(**写真 7**) の展示，デモを行っていた．同製品は，従来の使いやすい GUI はそのままに，ユーザーアクセス権限の設定など，さまざまな管理機能が強化されているという．

〔写真 7〕 **バックボーン・ソフトウェアの NetVault 7**

第141回ソフトウェア工学研究会
パターンワーキンググループ設立記念セミナー

■日時：2003年5月23日（金）
■場所：早稲田大学（東京都新宿区）

情報処理学会ソフトウェア工学研究会の主催により，ソフトウェアパターンの普及活動を行う「パターンワーキンググループ」の設立記念セミナーが開催された．

基調講演は建築家の中埜博氏（まちづくりカンパニー・シープネットワーク）．中埜氏は，パターンランゲージを提唱した建築家であるC.Alexander氏に師事し，パターンランゲージの建築への適用を推進している人物である．「まちづくり」という巨大なプロセスを実現するためには，住民と建築家の間での共通認識を構築するための言語＝パターンランゲージが重要になることや，パターンランゲージは単なるルール集ではなく評価基準であるということなどを述べた．

（株）東芝e-ソリューションの細谷竜一氏による「形から入らないパターン活動」は，組織内で独自のパターンを発見し，運用することを中心とした講演だった．パターンの「ライフサイクル」に注目し，「組織内だけで使用する段階」と「公開して改善を行う段階」を繰り返すことによりパターンを洗練させることの大切さなどを中心に扱った．

中埜　博氏

SANRAD社，iSCSIストレージ製品「iSCSI V Switch 3000」を発売

■日時：2003年5月27日（火）
■場所：大手町サンケイプラザ（東京都千代田区）

SANRAD社は，iSCSIストレージ製品「iSCSI V Switch 3000」を発売し，同時に（株）ネットマークスと（株）ネットワールドは，SANRAD社と日本国内における代理店契約を締結したことを発表した．

iSCSIは，SCSIパケットをIPパケットでカプセル化し，IPネットワークを介してSCSI機器を接続するための技術である．iSCSI V Switch 3000は，サーバとはiSCSI（ギガビットEthernet）で接続し，ストレージとはSCSIまたはファイバチャネルで接続，相互のプロトコル変換を行う「スイッチ機能」と，複数の物理ドライブを一つの論理ドライブとして認識させる「ストレージの仮想化機能」を提供する．iSCSIストレージはローカルドライブとして認識され，通常のドライブと同様のファイル操作が行えるほか，動的なドライブの追加などが可能になっている．価格は￥3,825,000～．

iSCSI V Switch 3000

EPSON，ホームネットワーク用コントローラ「S1S61000/S1S65000」を発売

セイコーエプソン（株）は，ディジタル家電などをネットワーク対応させるコントローラ「S1S61000/S1S65000」を発売した．

S1S61000はCPUコアとしてARM720Tを搭載し，TCP/IPプロトコルスタックを内蔵している．8/16ビットバスをもったCPUに接続することにより，ネットワーク機能を追加することができる．

S1S65000は，S1S61000にカメラインターフェースとJPEGエンコーダを追加したチップである．カメラモジュールを接続するだけで，ネットワークカメラを実現できる．

S1S61000

リネオ，統合クロス開発環境ELITEを発表

リネオソリューションズ（株）は，組み込みLinux向け統合クロス開発環境ELITEを発表を発表した．

ELITEはJavaベースのIDEで，Linuxカーネルの構築/アプリケーションの作成/デバッグといった一連の開発過程をLinuxおよびWindows上で行うことができる．SH/ARM/MIPS/PowerPCなどのCPUに対応している．

ハッカーの常識的見聞録

32

広畑由紀夫

今月の常識

FSB800MHzとその限界はいかに？

☆ Pentium4 HyperThread対応版FSB800MHzCPUが5月中旬より秋葉原に流れ始めました．「フライング発売か?!」といわれたこのCPUですが，どれほどの能力を秘めているのか，簡単にみてみることにしました．

秋葉原では，5月下旬にはASUS社P4P800シリーズなどi865系チップセットマザーボードまでが出荷され，「フライング発売ではない」という広報のコメントが出るほどの騒ぎになったi875/i865とFSB800MHz版Pentium4 CPUをさっそく入手し，テストしてみることにしました．

とくに今回は目玉となる，デュアルチャネルDDRとFSB800MHz，さらには，HyperThreadとの組み合わせでパフォーマンスが本当に上がるのかどうかをみてみました．

● デュアルチャネル

さて，筆者が入手したASUS P4C800(i875P)マザーでは，デュアルチャネルで動作しない場合に「どのような挙動を起こす」か？などを調査するため，あえてバルク品を購入しました．しかし，あっさりとデュアルチャネルで動作してしまいました．追加で購入した2枚の512Mバイトのメモリは同一メーカー/同一製品ではありますが，いずれの組み合わせでもメモリクロック333MHzでデュアルチャネル動作しました．気になるのは，オート設定では266MHzのデュアルチャネル動作になってしまうため，333MHzでBIOSなどの動作確認ができたとはいっても安心できない点です．

● FSB800MHz

FSB800MHzへの対応ということで，CPUへの基本クロックは100/133MHzから200MHzベースへと一気に上がりました．ノースブリッジとCPU間の転送が従来よりも高速化されたわけです．CPU自身やノースブリッジ間はここ数年高速化が進みましたが，CPUとノースブリッジ間の転送はPentium4登場以来ほとんど変わらずにきていました．そうしたなか，CPU内部のみが高速でも，メモリだけが高速でも，最終的につなぐパイプであるCPUとノースブリッジ間のFSBクロックの向上がなされたことは，CPUの処理能力が，CPUとメモリ間の転送で頭打ちになってきたのではないかと受け取ることができます．

● HyperThread

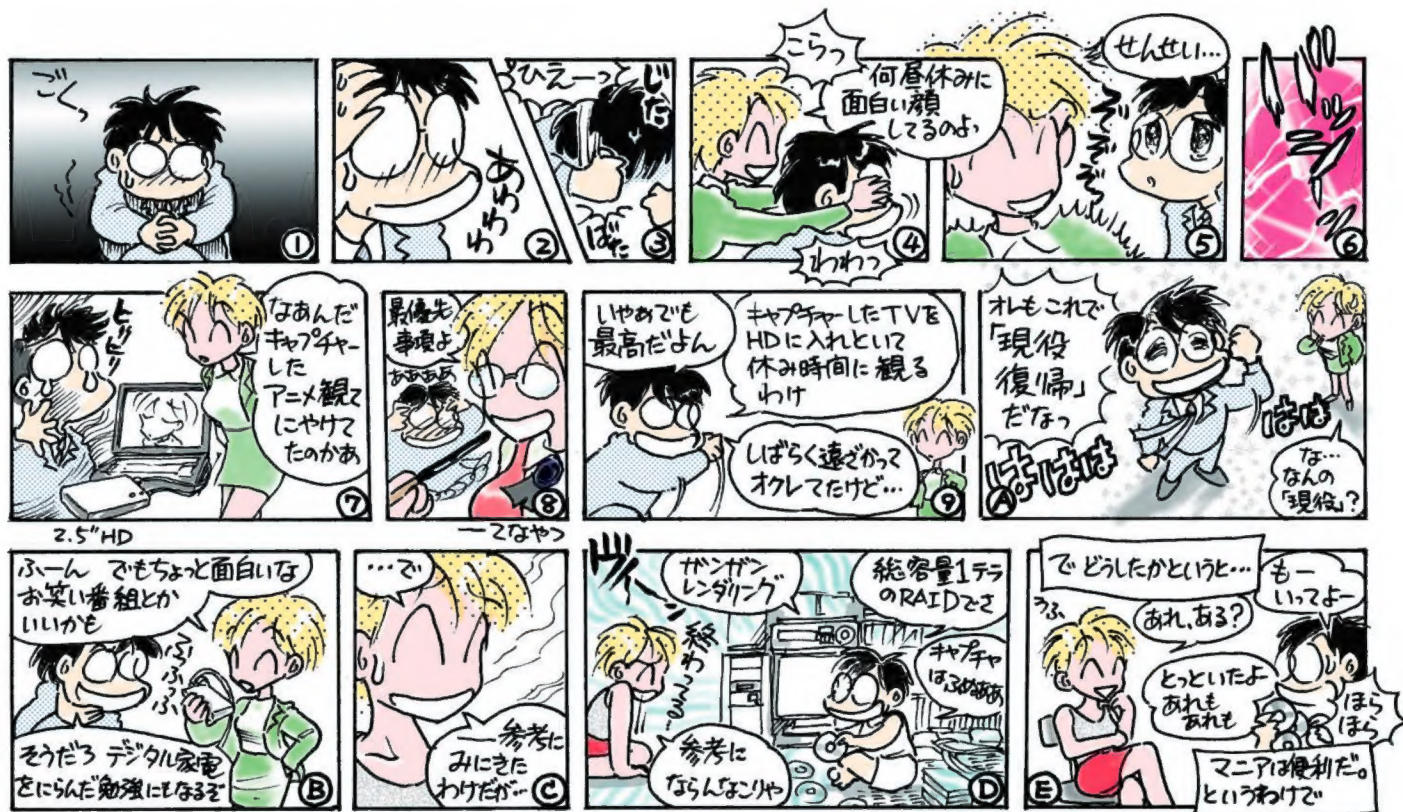
HyperThreadは，すでにFSB533MHzの3.06GHz Pentium4シリーズに実装されています．また，Xeon系ではそれ以前から実装されています．今回のFSB800MHzはCPUにおける計算能力の向上にともなって，CPUとノースブリッジ間の転送能力がさらに向上しています．さらにはデュアルチャネルDDR320MHz(FSB800MHz CPU使用時にはBIOSがDDRメモリクロックを320MHzに設定)に対応することで，ノースブリッジとメモリ間の転送がさらに速くなっているようです．こうすることで，HyperThread機能がより効率よく働き，大量のメモリ参照などを行うアプリケーション動作がより高速になっているようです．

● 危険な「賭け」に挑戦して限界をみてみる

昨年購入し，今も筆者のメインノートとして活躍しているTOSHIBA G6/U22は，2002年末のハイエンドノートクラスでFFXIが動作可能な機種として話題を集めたノートです．FFXI公式ベンチマーク「タルベンチ」はG6/U22の場合「2500～2800」ポイント程度です．この速度は公式ページの案内によると「通常のプレイに支障のない速度をもっている」とのことです．たしかにプレイしてみてコマ落ちなども比較的少なく，別売の専用コントローラを使用してもプレイに支障が出るような速度的な問題はありませんでした．

さて，FSB800MHzにGeForceFX5800を搭載し，デュアルチャネル320MHz，さらにはHyperThread対応版2.6GHzでの参考値は，なんと「5500～6000」ポイントでした．ここまでなら「新品で買ってきたらなんとかなる」ところですが，危険を承知でオーバクロックを煮詰めた状態にすると「6997」ポイントまでの向上が，暴走することなくみられました．このときのFSBクロックはすでに1GHzをこえて255MHz × 4です．AGPは10％のオーバクロックにとどめておき，他の設定もOSおよびドライバの設定調整程度で持ち上げることができました．ただ，これらの値は筆者の環境で行ったもので，同一メーカーの部品を使用しても，同程度の値が出ないこともありますし，より高い値が出ることもあるでしょう．

オーバスペック環境はたしかに「故障・破損・事故が起こっても保証はない」とはいえ，性能限界を調査し，使用している部品の性能的な余裕などを調べておくことは重要でしょう．今回出荷されているCPUおよびチップセットに関しては，筆者の簡単なテストでは余裕をもって製品化されていると判断することができそうです．そうした余裕があるからこそ安定し，安心して使えるということは重要だと思います．まして，今回のFSB800MHz対応CPUおよびチップセットは，従来の壁を一つ乗り越えた速度をもっているといえそうです．

ひろはた・ゆきお　OpenLab.

フジワラヒロタツの現場検証(71)

マイブーム

さすがにずっとこの分野に関わっていると，やれ設計だ，やれ現調だ，やれ火消しだ(!?)と，同じようなことの繰り返しにいささか倦んでくるものです．

この傾向というものは，ここ数年来，密やかに通奏低音のごとく耳の奥底に響いていたのですが，多忙な生活が，ある意味気を逸らしていてもくれたようで，表面上は平穏に(？)せわしない日々を送っておりました．それがこの頃，どうもさまざまな疲れが噴出してきたようで，すっかり精神的に参ってしまったなあと，声に出してみると，そんなに参ったふうではないものの，やはりあまり元気ともいい難いのです．どうもいけません．

生き馬の眼を抜くようなIT業界．はからずもそのすみっこのおこぼれで生計を立てている筆者としては，こんなことでは商売に差し支えます．何とかしなくてはと考えておりました．

ところが，そんな憂鬱な日々をすごしていた筆者にある日，転機が訪れました．最近話題のDVDビデオやらキャプチャやらにはまってしまったのです！

そもそも筆者は，自分の計算機環境にだいたい満足しておりました．Pentium4の1.6GHz，256Mバイトのメモリ，80Gバイトのハードディスクにタブレット——本誌のマンガを描くくらいであれば，それで十分なのです．会社では，Pentium IIマシンを，ぶつぶついいながらも使っています．ドライバ屋にそんなに新しいマシンは必要ありませんし，少し枯れたくらいのほうがいい場合もありますよね．新しいマシンが欲しいなどとこぼしながらも，それはそれで，まあ満足しておりました．

ところが，ふとしたきっかけでパソコンでのビデオ録画を始めてしまったというわけです．1万円程度のチューナ付きのビデオキャプチャカードを購入し，それで最近放映中のアニメや落語などを録り始めたのです．

するとどうでしょう，アッという間にHDは足りなくなる！ さらに記録型DVDを買いに走り，DVD-Rではなく，DVD+Rを購入して，リビングのDVDプレーヤで読めずに情けない気持ちになる(笑)！ 果てはあれほど満足していたパソコンの処理速度が，ビデオ圧縮のために不満に思えてくる，などなど！……．すっかり計算機環境のリフレッシュへの欲望とともに，おたくな趣味も盛り返しました．「おた」の「ぶり返し」です．

時間がないからと見逃していたさまざまなアニメや，NHKの落語番組やらを，昼休みに1話ずつ，アニメはその気恥ずかしさに身悶えしながら，こつこつと追いはじめ……．

あらためて，筆者の「原点」というものは，こういった「自分に」役立つシステムをあれこれ構想し，組み上げていくことだったなあと気づきました．その過程で得た知識や経験が飯の種になっていました(そして，かつてはそれがデバイス中心だった)．最近そういったことを思い返しながら少しずつ「やる気」をとり戻しています．とても財布が軽くなるのが困りものなのですけれども，なかなか納得できる自己投資なのでした．

藤原弘達　(株)JFP　エンジニア，漫画家

特集 動きのはやい技術動向をより効率的に理解するための用語解説

現代コンピュータ技術の基礎

　最近のコンピュータ/エレクトロニクス技術は，進歩がとても速く，新しい技術が次々に開発されてくるうえに，細分化も進み，技術の動向を追うだけでもたいへんです．開発現場のエンジニアは，自分の専門以外の分野だと，内容をなかなか理解できないこともあります．

　そこで今回の特集は，最近の本誌の特集記事（2002年8月号～2003年4月号）の中に出てきた，さまざまな用語について，「用語集」という切り口から，技術解説を行います．本誌特集は，その時点における重要技術を掘り下げて解説しており，その用語を理解することは，ひいてはコンピュータ/エレクトロニクス技術の流れを理解する「早道」ともなります．座右に置いて，役立てていただけると幸いです．

（編集部）

さまざまな技術が使われる基盤となる

組み込みシステム開発の基礎知識

宮原淳一

用語解説特集の導入として，さまざまな技術が集約されるインフラとしての「組み込みシステム」の基礎を解説する．まず，組み込みシステムがどういうもので，どんな特徴をもつものかを解説する．そして，組み込みシステムを開発する環境の「いままで・これから」を，課題やさまざまなトピックも含めて，ていねいに解説する．

（編集部）

はじめに

携帯電話，ディジタルカメラ，プリンタなど，多くの身近な機器にはコンピュータが組み込まれています．このような機器のことを**組み込みシステム**と呼びます．組み込みシステムにおけるコンピュータは，メインフレームやミニコンピュータ，パーソナルコンピュータ（PC）などがもつ「普通」のコンピュータとしての機能を提供するのではなく，各機器がもつ機能注1の実現を，目に見えないところで強力にサポートしています．

そこで，このような機器ごとに異なる使用方法を提供する組み込みシステムは一体，どのような開発環境で開発されていくのか，それに使用されるソフトウェア開発の基礎知識を中心に解説します．

〔図1〕組み込みシステムの例

1 組み込みシステム

組み込みシステムの例を**図1**に示します．

半導体技術の進歩にともない，安価で高機能なコンピュータが提供されるようになりました．その結果，多くの機器にコンピュータが組み込まれるようになり，機器の付加価値を増大させています．今日では，現代社会を支えるもっとも重要なインフラになっているといっても過言ではありません．

1.1 組み込みシステムの特徴

組み込みシステムは非常に幅広い機器で使用されるため，その性質も分野により異なりますが，一般的に組み込みシステムの特徴としては，次のようなことがあげられます．

● **ハードウェアリソースの制約**

組み込みシステムは，とくに大量生産される場合，コストに対する要求が厳しいため，ハードウェアリソースがしばしば制限されます．これによりソフトウェアには**表1**のような制約が課せられます．

● **応答時間（リアルタイム性）**

組み込みシステムでもっとも考慮しなければならない要件が**応答時間**です．**リアルタイム性**ともいいます．ほとんどの組み込みシステムでは，何らかのイベント（システムに応答を要求する事態）が発生すると，それに対して一定時間内に何らかの応答を返すことを要求されます．さまざまなイベントに対して制限された時間内に応答するためには，システム内にそれに即したしくみが必要となります．このようなシステムを**リアルタイムシステム**と呼びます．

大半の組み込みシステムではリアルタイム処理を必要とします．たとえば，自動車を走行中，ハンドル操作を誤り電柱に激突したとします．その場合，激突したというイベントが発生してから，制限時間内に応答としてエアーバッグが開かなければ，命

注1：携帯電話は会話，メールの送受信やインターネット接続，ディジタルカメラは画像処理，プリンタは紙面への印刷といったように．

〔表1〕ハードウェアによる制限

ハードウェア	該当ハードウェアがプログラムに課す制限
CPU	処理速度：低速CPUかつ許容処理時間が短い場合，同じ機能モジュールをできるだけ少ないステップ数でプログラミングする必要がある
メモリ	プログラムサイズ：限定された容量のメモリにプログラムを格納する必要がある
タイマ	処理速度：一定周期で発生するタイマ割り込みを処理するプログラムは頻繁に実行されるため，処理時間を最小限に抑えないと，システムに負荷がかかる

にかかわる最悪の事態を招くでしょう．携帯電話の電話帳にアドレスを登録している最中に，電話がかかるというイベントが発生したにもかかわらず，応答として着信音が鳴らなければ，相手はしびれを切らして，通話を断念するかもしれません(**図2**)．

● **非常に高い信頼性・安定性**

組み込みシステムでは，長時間ノンストップで使っても故障しないことはもちろん，PCのようにハングアップしないことが要求とされます．たとえば，情報家電では不特定多数の人間による誤操作に対しても正常に動作することが求められます．また，分野(航空機や自動車関連など)によってはフェイルセーフ性(故障時安全性)やフォールトトレラント性(耐故障性)がきわめて重要視される場合があります．

1.2 リアルタイムOS (RTOS : Real-Time Operating System)

組み込みシステムの多くには，リアルタイム性を保証するしくみをもったOS(オペレーティングシステム)が使用されています．このようなOSを**リアルタイムOS**(**RTOS**)と呼びます．RTOSはOSが一般的に提供する機能要求を満たすだけではなく，多くの組み込みシステムに要求される厳しい時間制約の中で，効率よく処理するしくみ(リアルタイム処理機能)も提供する必要があります．

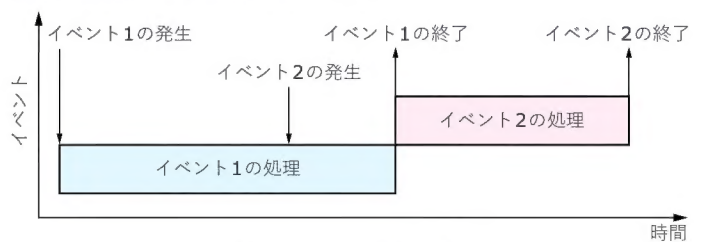
リアルタイム処理機能を司るソフトウェアが，RTOSの中核である**カーネル**です．カーネルのリアルタイム処理機能の良し悪しが，組み込みシステムの性能を大きく左右する場合があります．

● **リアルタイム処理機能**

カーネルは，アプリケーションが制限時間内に応答できるしくみを提供します．発生するイベントが一つの場合，すぐに応答が可能です．その応答時間が制限時間内に収まりきれないのであれば，ハードウェアの改善などを行うしかありません．ところが，あるイベントの処理を行っている最中に，別のイベントが発生することがあります．システムはどちらのイベントを優先して処理するかを決定しなければなりません．その決定処理を**スケジューリング**といいます．**カーネル**は，このスケジューリング機能を提供しています．

図3(a)では，イベント2が発生してもイベント1の処理を実行し続け，イベント1の処理が終了後，イベント2の処理を開始しています．逆に**図3(b)**では，イベント2が発生すると同時にイベント2の処理を優先的に実行し，イベント2の処理が終了後，中断されたイベント1の処理の続きを開始しています．

〔図2〕電話がつながらない

〔図3〕複数イベントのスケジューリング

(a) イベント1の処理を優先

(b) イベント2の処理を優先

このように，カーネルのスケジューリング機能により，どの処理を優先的に実行させるかを決定する指標を**優先度**といいます．優先度が高いイベントが優先的に実行されます．

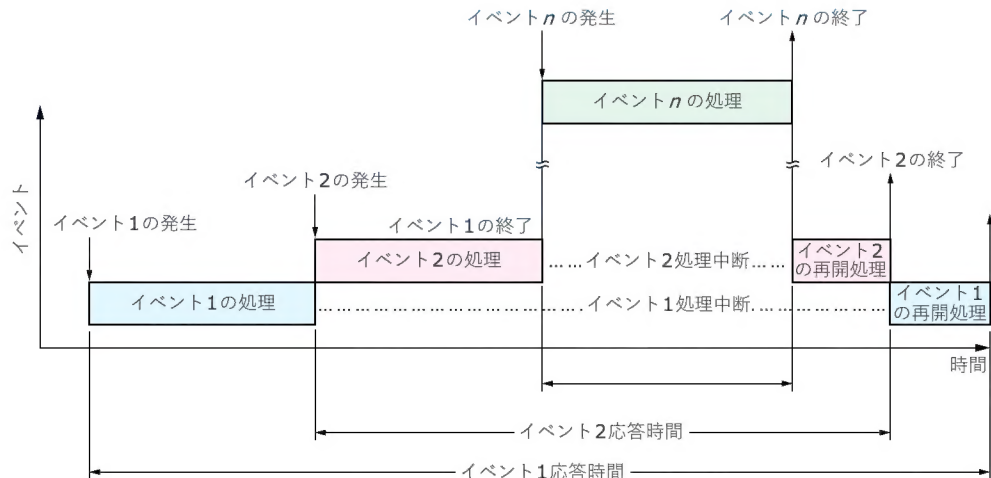
次に，優先度の低い順に複数のイベントが発生した場合の処理について，説明します．

図4は，もっともわかりやすい複数イベントの処理の流れを示したものです．ここでは，イベントの発生順の問題を取り上げているわけではなく，次のことを理解してください．

① 最優先度のイベントの応答時間＝そのイベントの処理に必要な時間

② ある優先度のイベントの応答時間＝そのイベントの処理に必要な時間＋自分より高優先度のイベントの処理に必要な時間

組み込みシステムは，すべてのイベントの応答時間に制限があ

〔図4〕各イベントの応答時間

ります．**図4**から読み取れるように，応答の制限時間が短いイベントほど高優先度で処理することは合理的であるといえます．

● **RTOSの選択**

RTOSを用いるだけでリアルタイムシステムを構築できるものではありません．アプリケーション側でもリアルタイム性を考慮した設計を行うことにより，初めてリアルタイムシステムを構築できるのです．構築したい組み込みシステムは，非常に厳しい制限時間内での応答処理が要求されるのか，比較的緩やかな応答時間を保証すればいいのか，という点を考慮し，使用するRTOSの性能をチェックする必要があるでしょう．さらに，使用するCPUでの採用実績は豊富か？　使用可能なソフトウェア部品はそろっているか？　ライセンスフィーはどうか？　など，あらゆる判断基準をもとに搭載するRTOSの選定をする必要があります．

2 組み込みシステム開発環境の遷移

2.1 半導体技術

半導体技術の革新的な進歩を抜きにして，組み込みシステム開発環境の遷移は語ることができません．1947年にトランジスタが，1957年にICが発明されて以来，一貫して微細化，高速化を成し遂げています．半導体の集積度はほぼ1年半で2倍といわれており，今日では，10億個以上のトランジスタの集積を可能にしています．

こういった革新的な進歩のもと，ここ数年，登場した考え方がSoC（System on Chip）です．SoCは従来のようなプリント基板の上にCPU，ROM，RAM，ASSP（Application Specific Standard Product）といった半導体を並べてシステムを構成するのではなく，一つの半導体チップ上にシステムにとって必要な機能をすべて実現させるものです（**図5**）．このSoCが脚光を浴びるようになってきた背景の一因として，携帯電話などの高性能なモバイル機器が急速に伸びていることがあげられるでしょう．

マイクロプロセッサ，チップセット，ビデオチップ，メモリなどの機能が1チップに集積されることにより，実装に必要な面積が劇的に縮小し，同等の機能をもつ複数チップによるシステムと比べて消費電力も格段に抑えることが可能になります．小型化し，省電力が求められるモバイル機器にはうってつけの技術といえます．では，このような半導体技術の進歩にともない，組み込みシステムを構成する主要な要素である，「ソフトウェア」の開発手法はどのように遷移したのでしょう．

〔図5〕SoC

（a）従来のプリント基板

（b）一つのLSI上に実装

2.2 組み込みソフトウェア

組み込みシステムで使用されるソフトウェアを**組み込みソフトウェア**と呼びます．一般的に，組み込みソフトウェアの開発はWindowsアプリケーションのようなPC上のソフトウェア開発とは異なり，**クロス開発環境**で行われます．クロス開発環境とは，ホストマシン（プログラムのコンパイルやリンクを行うマシン）とターゲットマシン（コンパイルされたプログラムを実行するマシン）が異なる開発環境のことを指します（**図6**）．

2.3 SoC出現前の組み込みソフトウェア開発手法

SoCのような高集積，高速化された半導体チップが出現する以前，開発環境として重宝され続けてきた環境が，ICEを使用した開発です（**図7**）．ICEとはターゲット上のCPUの実装される場所にプローブ（CPUが内蔵されたもの）を当て，CPUの代わりに動作をエミュレーションしてくれるのと同時に，デバッガと連動して組み込みソフトウェアの動作検証が可能な装置の

〔図6〕クロス開発環境

〔図7〕ICEを使用した開発環境

〔図8〕JTAGエミュレータ

ことです．

ICEには外部メモリ(ROM/RAMなど)を代替するエミュレーションメモリ機能があり，プログラムのダウンロードや変更などが容易です．とくに，外部バス情報をリアルタイムに検証するリアルタイムトレース機能はプログラムのデバッグには欠かせないものであり，探すのが困難なバグを瞬時に発見することも可能です．開発初期段階の周辺デバイスとのインターフェースの整合性の調査，デバイスドライバ作成時の不具合発見などには有効です．

しかし，周辺回路やIPなどがユーザーでカスタマイズされたSoCではCPUのコントロール信号が内部に集約されてしまうため，SoCの外部ピンをプロービングしてもCPUのエミュレーションはできません．こうした周辺回路の統合，またはCPU動作周波数の高速化などにより，SoCをICEでデバッグするためには，ICEを意識してチップを開発する必要があるうえに，ICEはすべて特注品となり，製品のTime-To-Marketの実現，開発費の削減が必須となってきた今日では，ある意味限界に達した感がある開発環境といえるでしょう．

2.4 SoC出現後の組み込みソフトウェア開発手法

CPUのコントロール信号が外部に出ないSoCや，高集積化されたがゆえに，配置されるピン間が非常に狭くなり物理的にプローブを接続するのが困難になってきた半導体チップ上のソフトウェアの開発手法として注目されてきたのが，**OCD**(**On Chip Debug**)です．OCDとは，CPUにデバッグ機能を埋め込む手法です．OCDはBDM(Background Debug Monitor)やUDI(User Debug Interface)，JTAG(Joint Test Action Group)など，半導体チップを供給するメーカーにより方式が異なります．

その中でも主流は，**JTAG**を利用する方法です．元来，JTAGはIEEE 1149.1という規格を推進したグループの名称ですが，今では規格名よりグループ名で広く普及しています．この規格では，**バウンダリスキャン・アーキテクチャ**とこの機能に外部からアクセスするためのシリアルポートの仕様を規定しています．

バウンダリスキャン・アーキテクチャとは，ターゲットとなるICとデータのやりとりをするためのアーキテクチャです．IC内部のコアロジックと各ピンの間に，テストプローブと等価な働きをする「セル」と呼ばれるレジスタを配置し，これを結合してシフトレジスタを構成，このシフトレジスタを制御することにより，テストコードの入力とこれに対する応答によりテストを実行していきます．このように，デバイスの内部と外部の境界をスキャニングすることから，バウンダリスキャン・アーキテクチャと呼ばれます．

JTAGエミュレータはバウンダリスキャンを用いて，CPU内にあるデバッグ機能にアクセスします(**図8**)．従来のICEとは異なり，実際のCPUを使用するため，プローブを着けたときに信号タイミングがずれることや，消費電力が変わるなどということがありません．また，プローブをつける必要がないため，取り扱いが楽になります．

しかしながら，JTAGエミュレータは従来のICEに近い機能は提供できますが，エミュレーションメモリが存在しないため，ターゲットの実メモリ上にプログラムをダウンロードする必要があります．また，リアルタイムトレース用のメモリも存在しないため，リアルタイムトレースができないなどの制限もあります．

2.5 その他の組み込みソフトウェア開発手法

ICEやJTAGエミュレータが出現する前から現在に至るまで，長年，組み込みソフトウェア開発において，使用され続けている開発手法を二つ紹介します．一つは，**ROMモニタ**を利用した手法です．ICE用のICソケットやJTAGエミュレータといっ

たデバッグを行うためのハードウェア的なしかけがないターゲットCPU上でもソフトウェアの動作検証を可能にするために，ターゲットボードに実装されているROMにデバッグモニタと呼ばれるプログラムを書き込んでおき，そのプログラムが開発ホストとシリアルやパラレル経由で通信してコマンドを受け取り，プログラムの実行を制御することができます．

ROMモニタは，ICEやJTAGエミュレータのようにハードウェア的なデバッグのしくみを提供するのではなく，プログラム，すなわち，ソフトウェア的なデバッグのしくみを提供します(**図9**)．そのため，ハードウェアブレークや外部バス情報を取得するリアルタイムトレースなど，ハードウェア的な機能の実現が困難な場合がありますが，ICEやJTAGエミュレータで使用する専用のボックスが不要であり，製品化後の保守デバッグが容易になるなど，さまざまな応用的な機能を比較的簡単に実装できます．しかしROMモニタは，メモリが内蔵されたチップでは物理的に無力になってしまい，使用できなくなります．

二つ目が**ISS**(**Instruction Set Simulator**)です．ISSは，ターゲットマシンがなくても，ホストマシン上で，ターゲットCPUがもつ命令セットを疑似的に実行できるソフトウェアです(**図10**)．

ISSは，CPUがもつパイプライン処理を正確に行うものから，クロックの精度が実システムの±数～数十%程度の誤差で済むようなものまであります．当然，ISSはソフトウェアで実現するので，実CPUと同等の精度に近づければ近づけるほど，処理速度は落ちます．精度と速度はトレードオフの関係になります．また，命令セットを疑似的に動作させるだけではなく，周辺デバイス(タイマ，割り込み)のシミュレーションを行うような高機能なISSもあります．

ISSを採用するメリットとしては，ターゲットマシンがなくてもソフトウェアの開発を行えることでしょう．とくに組み込みシステムの世界では，コスト要求が非常に厳しいため，試作機の作成は必要最小限に抑えなければならず，またハードウェアの不具合・リリースの遅れにより，ソフトウェア開発が遅れてしまうことは多々あります．それらを解消するためには，ISSを使用した先行開発手法は有用でしょう．

しかし，しょせん「ソフトウェアエミュレーション」なので，実ターゲットとまったく同等の動作を期待することはできません．使い方しだいです．

ここまで組み込み開発環境の遷移について，開発手法にスポットを当てて述べてきましたが，どの手法にも一長一短があります．ただ，半導体技術の革新的な進歩に，今日までの組み込みソフトウェアの開発手法が追いついていないように思えます．そのせいか，新しい開発手法を望んでいる声を開発現場からよく耳にします．

3 組み込みシステム開発環境の現状

急速な技術革新により生まれる新しい製品は，年々世の中への普及が早まる傾向にあります．

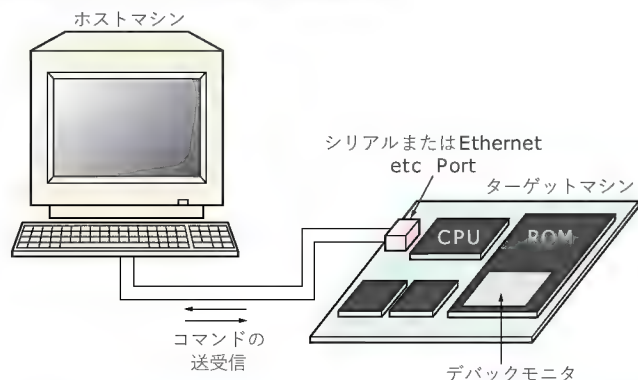
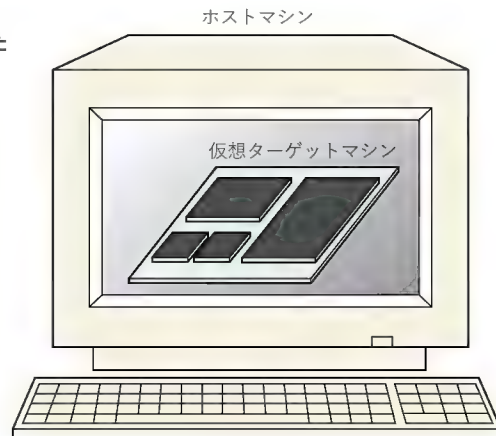
図11に示されるように，たとえば白黒TVとビデオや携帯電話などを比較すると，出荷台数が100万台に達する時間は大幅に短縮されています．2000年に発売されたPlayStation 2は，わずか2日間で100万台を出荷しました．この事実が示すとおり，非常に短期間で製品を出荷してマーケットシェアを獲得することがいかに重要かというのがわかると思います．製品出荷が遅れ，後手を踏んでしまうはめになると，出荷台数も限られ，売り上げも上がらず，利益も十分獲得できなくなるということになります．

また，組み込みシステムの高機能化や高度なGUI，そして，ネットワーク化による機能の急増により，組み込みソフトウェアの大規模化・複雑化も非常に著しくなっています．

このように，現在の組み込みシステム開発では，Time to Marketの短縮，ソフトウェアの大規模化，複雑化に対する要求から，ソフトウェアの生産性の向上が課題となっているのが現状です．

3.1 組み込みシステム開発の悩みの種

RTOSの使用が一般的になった組み込みシステム開発において，もっとも頭を抱える悩みは，組み込みシステム技術者の不

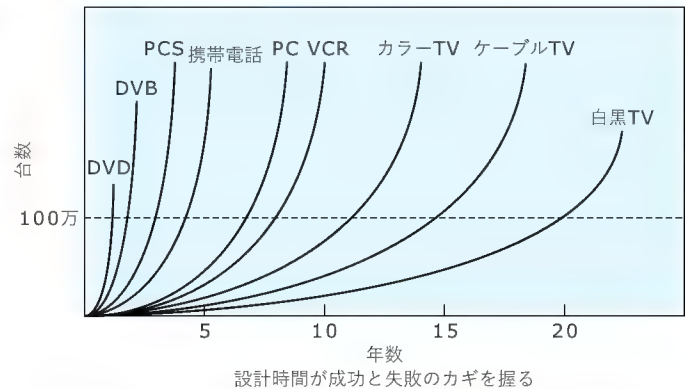
〔図9〕ROMモニタを使用した開発環境

〔図10〕ISSを使用した開発環境

足です．(社)トロン協会が2000年11月に行った「組込みシステムにおけるRTOSシステムの利用動向とITRON仕様OSに関するアンケート調査結果」の中で，「RTOSの問題点」の項を見てみると，1位：技術者不足(32.3％)(**図12**)となっています．

ではなぜ，組み込み技術者が不足する(＝育たない)のでしょうか？　組み込みシステムはリアルタイム性やハードウェアリソースなどの制約が多かったり，使用するターゲットが異なったりなど，そもそも組み込みソフトウェアのプログラミング自体，難しいものです．その原因を突き詰めていくと，「RTOSの問題点」の別原因にもなっていますが，4位：開発環境/ツールの不足(9.0％)が大きな一因だと筆者は考えています．開発環境・開発ツールが不足していることにより，よくいうとスーパー職人芸，悪くいうと，その場しのぎのプログラミング手法に頼らざるを得なくなり，その結果，一般的な開発手法が根付かずに組み込み技術者が育たないのではと考えています．

3.2　組み込みシステム開発手法の変革期

一昔前，4ビット，8ビットCPUを使用した組み込み機器が主流だった時代は，熟練した技術者による独自の開発ツールおよび独自の開発手法で間に合ってきました．ところが今では，16ビット，32ビットもしくは64ビットCPUの世界に移行していくと同時に，システムの大規模化，Time To Marketの短縮にともない，いままでの少人数による場当たり的なソフトウェア開発手法では限界となっています．といって，スキルが不足している技術者をポンと開発現場に投入できるかといわれれば，会社(プロジェクトマネージャ)側としては，それによるコストや工数の増加は避けたいでしょう．

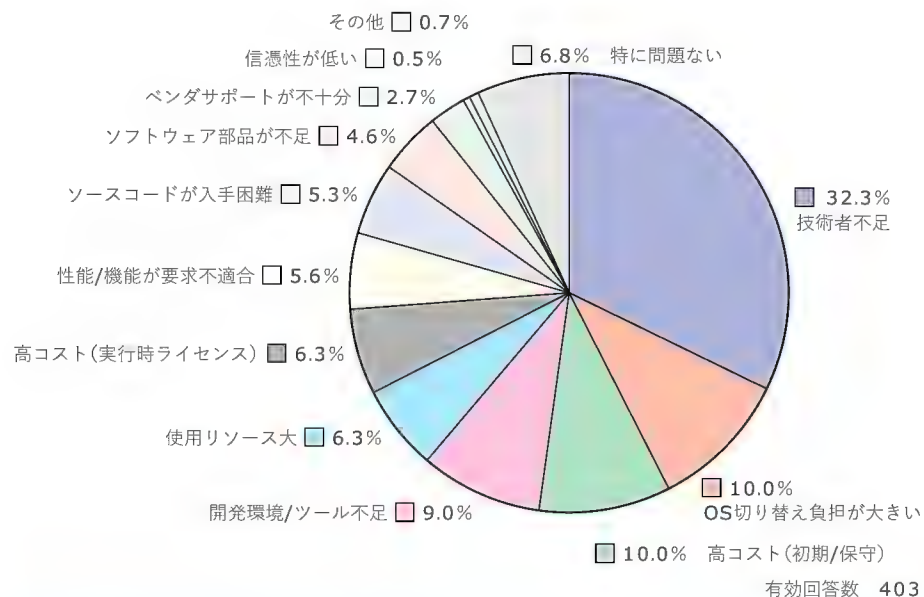
しかし，そうもいっていられません．できるかぎり多くの技術者を開発現場へ投入し，市場競争に勝てる製品をいかに短期間で開発するかが重要です．そのためには，開発現場に，ソフトウェア開発に最適な開発ツール，統一化された開発手法を導入することが必要ではないでしょうか．

〔**図11**〕**Time to Volume**

出典：Synopsys;D.Merrman,"Wireless Communications t Report," BIS, Boston, 1995;Dataquest

では，このような問題を解決できる理想的な開発環境は一体あるのでしょうか？　残念ながら，存在しているとはいえません．しかし，開発手法の変革期である今日，一歩一歩着実にそれに近づきつつある開発環境がいくつか存在していることは確かなようです．

4　次世代の組み込みシステム開発環境

組み込みシステムで使用されるソフトウェア部品は，組み込みシステムの特性上，大きく分けて，二つの部品で構成されると考えられます．**カスタム部品**と**標準部品**です(**図13**)．カスタム部品とは，その機器のユニークな機能を実現するためのソフトウェア部品です．競合他社製品と差別化するうえで欠かすこ

〔**図12**〕**RTOSの問題点**

とのできない部品です．一方，標準部品とは，何らかの標準規格に準拠したソフトウェア部品のことを指します．TCP/IPプロトコルスタックやファイルシステムなどがその部類に入ります．競合他社製品と差別化まではいかないまでも，一般的にあるべき機能，いわば枯れたような機能が必要な場合，スクラッチから開発するより，標準部品を外部から調達したほうが，開発期間・コストなどを考慮すると，望ましい場合が多々あります．

組み込みソフトウェアの大規模化・複雑化にともない，ソフトウェアの生産性の向上が今日の課題となっているものの，従来の開発手法では，カスタム部品であれ，標準部品であれ，個々のソフトウェア部品で構成されるシステム全体を把握し，検証することが難しいのは想像に難くないでしょう．

それゆえに，今後の開発ツールおよび開発手法に求められる要件は，ソフトウェア部品に重きを置いたシステム構築・デバッグ環境を整備することではないでしょうか．そのためには，開発ツールはもちろんのこと，ハードウェア，ソフトウェア両面からの力強いサポートが必要不可欠です(**図14**)．

次に，筆者が考える，今後の組み込みシステム開発環境に求められる要素を挙げてみます．

① 高機能かつ利便性のある開発ツールの汎用化
② 付加価値をもつソフトウェア部品の再利用と技術の共有化
③ ターゲット環境の共通化

4.1 高機能かつ利便性のある開発ツールの汎用化

従来の組み込みソフトウェア開発ツールは，プログラムの実行制御やトレース機能，計測・データ解析など，基本的なデバッグ機能の提供で手一杯でした．そこで，今後の開発ツールでは，次のようなデバッグ機能を充実させる必要があると考えています．

① 洗練されたマルチプログラミング環境
② ソフトウェア部品固有のデータ構造を取得するシステムスナップショット
③ システム内で発生した各種イベントの情報の履歴取得および解析

● 洗練されたマルチプログラミング環境

リアルタイム処理を実現するために，複数のプログラムがイベントに応じて切り替えが可能になるよう作成されている必要があります．そのプログラム制御のことを**マルチプログラミング**と呼びます．マルチプログラミング環境で実行される逐次処理単位を一般的にタスクと呼びます(OSにより，スレッド，プロセスとも呼ぶ場合がある)．マルチプログラミング環境のことをマルチタスク環境とも呼ぶことができます．

リアルタイムシステムを構築するうえで，「良いタスク」を作成することが非常に重要です．マルチタスク環境では，基本的に，自タスクはいつ他のタスクや割り込みの発生により処理が中断されるかわかりません．カーネルの排他制御機能を利用して中断させないようにすることは可能ですが，排他制御を多用するとシステム全体の即応性が低下したり，処理優先度の逆転などが発生するため，極力最小限の使用に留める必要があります(**図15**)．このような要件が十分に考慮，検証されていることが「良いタスク」の条件です．

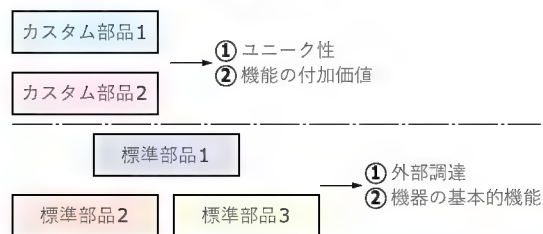
ところが，自分で作成したタスクが「良いタスク」かどうかを検証するのは容易ではありません．ICEなどの従来の開発ツールでは，割り込みを含めたシステム全体を停止させ，ステップ実行などの実行制御を行わざるを得なくなり，実システム(リアルタイムシステム)の動作と異なった環境での検証となります．

そこで，システム全体を停止することなくタスクの検証が行えるような環境の整備が必要です．さらに，タスク間の同期を検証できるよう，複数のタスクを同時にデバッグ対象にすることも求められるでしょう．このようなマルチタスクデバッグ環境を開発ツール側で提供することにより，高品質なタスクを自然に作成することができるようになります(**図16**)．

● ソフトウェア部品固有のデータ構造を取得するシステムスナップショット

今日の組み込みシステム，とくにコンシューマ機器分野などでは，TCP/IPプロトコルスタックやWebブラウザ，またJavaアプリケーションを動作させるためのVMなど，内部構造が複雑なソフトウェア部品を搭載することが多くなってきています．

そこで，複雑な構造をもつソフトウェア部品をブラックボックス化したまま，データを瞬時に取得できるシステムスナップショット機能が求められるでしょう．このシステムスナップショット機能により，対象のソフトウェア部品の直接の設計者

〔図13〕カスタム部品と標準部品

〔図14〕
開発環境により，システムを構成する各ソフトウェア部品をサポート

※ソフトウェア部品一つ一つに重きをおいた開発環境が必要

や実装者以外の技術者が，そのソフトウェア部品を利用することが容易になります（**図17**）．

- **システム内で発生した各種イベントの情報の履歴取得および解析**

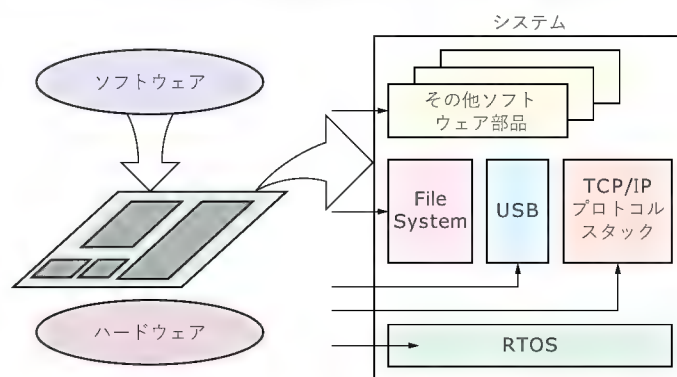
カーネルやミドルウェア，またユーザーアプリケーションからはさまざまなイベントが発生します．

- タスク/割り込みハンドラからの各ソフトウェア部品APIの呼び出しイベント
- カーネルのサービスコール，タスク切り替え，割り込み発生イベント
- ユーザーアプリケーションからの特定イベント

タスクの実行状態，割り込みの発生状況など，システムの実行遷移状態を確実に追跡するためには，これら各種イベントを

〔**図15**〕**排他制御による優先度逆転**

〔**図16**〕**マルチタスクデバッグ**

〔**図17**〕**システムスナップショット**

詳細にロギングし，さまざまな切り口で解析できる機能が必要になります．このような機能は，複雑なマルチタスク環境，およびリアルタイムシステムの不具合解析やボトルネックになっている部分を調査するためには非常に有用でしょう(**図18**)．

いずれの機能もすべて，システムを止めずに使用できることが大きな特徴です．リアルタイムシステムを構築するうえで，実際に動作しているシステムと同等な環境を初期の開発フェーズから用意できることは非常に有用です．ランニングテストなどの最終検証でプログラムを走らせた結果，複数のタスクが協調動作しなかった，設計した範囲内でタスクが正しいふるまいをしなかったなどが発見されるようでは，納期に支障をきたす場合が多々あるでしょう(実際の現場では，納期に支障をきたさないように「不夜城化」するわけだが……)．開発ツール側にはそのような問題点を解決することが求められます．

今日，上記に近い機能を付加価値として提供している代表的な開発ツールとして，Tornado(ウインドリバー製：http://www.windriver.com/japan/products/family/ide.html)やeBinder(イーソル製：http://www.esol.co.jp/embedded/index_ebinder.html)が挙げられます．

ところが両者とも，高機能で利便性をもった開発ツールではありますが，汎用的に使用できないのが現状です．特定のRTOS，そしてそのRTOS上でしか動作しないソフトウェア部品しか扱えません．さまざまなRTOS，ソフトウェア部品が共通で使用可能になれば，開発現場としては，同じ開発ツールを使用し，統一化された開発手法のもとに，スキルが不足している技術者も含めたすべての技術者をプロジェクトに投入することができるでしょう．

4.2 付加価値をもつソフトウェア部品の再利用と技術の共有化

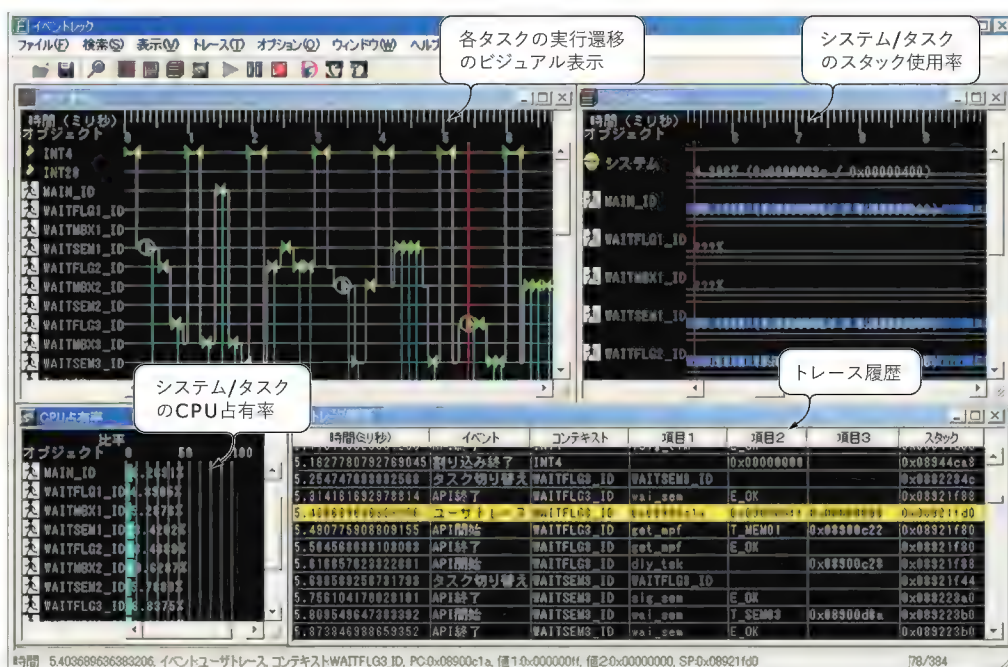
従来の組み込みシステムの開発環境では，標準部品を外部から調達してきたものの，使用するまでに手間がかかる，また，社内で作成したカスタム部品を特定部所，あるいは関連プロジェクトで共通で使用したいが，うまく流通しきれないという問題が根強く残っています．

今日の課題としては，ソフトウェア部品の再利用および，その部品を扱うための技術の共有化をいかに促進させるかを考える必要があります．

この課題に対してチャレンジしている手法の一例として，eBinderが提供するソフトウェア部品のパッケージ化支援機能があります．パッケージ化支援機能によりパッケージ化されたソフトウェア部品を「パーツパッケージ」と呼びます(**図19**)．パーツパッケージには，ソースコードやドキュメントのほかに，それらを扱うための各種情報が含まれています．各種情報は，パッケージ情報とリソース情報に大きく分かれます．

パッケージ情報として，その部品のベンダ情報，バージョン情報，また他のパーツパッケージとの依存・包括関係情報などが含まれています．それにより，たとえば，あるRTOS環境で使用可能かどうかを自動でチェックすることができます．もし使用不可能であれば差分情報を追加し，その環境で使用可能なパーツパッケージとしてデータベースに新規登録ができます．そして，データベースに登録したパーツパッケージをプロジェクトに追加することにより，そのプロジェクト内で使用(コンフィグレーション・ビルドなど)できるようになります(**図20**)．

一方，リソース情報としては，コンフィグレーション(設定)

〔**図18**〕システムイベント解析

情報，ビルド(構造)情報，デバッグ関連情報(管理情報，API情報)で構成されています．リソース情報およびソースコードをリファレンスすることにより，eBinder上から，対象のソフトウェア部品のコンフィグレーション，ビルド・デバッグを行うことができます(**図21**，次頁)．

パッケージ化を行うことによるメリットとして，次のような点があげられます．

① ソフトウェア部品の再利用の促進

パーツパッケージには，そのソフトウェア部品のバージョン情報，他のソフトウェア部品との依存・包括関係情報が含まれているため，管理がしやすくなり，その結果，他システムへの流用を比較的容易に行えます．

② ファイル構成やビルド方法を意識せずに対象のソフトウェア部品が使用可能

パーツパッケージは，プロジェクトに追加するために必要なソースファイルやビルド設定情報などを含んでいるため，プロジェクトへの追加が簡単(削除も容易)に行うことができます．

③ 効率化されたデバッグ手法の提供

ソフトウェア部品をデバッグする際には，本来ならば，自分で解析する(ソースコードを追いかけるなど)必要のあるソフトウェア部品の実装情報(内部データ構造，動作仕様)を詳細に知らなくてもシステム検証を行えます．

しかし，パーツパッケージ化という概念が世間一般に広がっているとはいえません．それが必要だと認識することは可能かもしれませんが，すべての開発現場において有用な機能かどうかは，議論の余地があるでしょう．

4.3　ターゲット環境の共通化

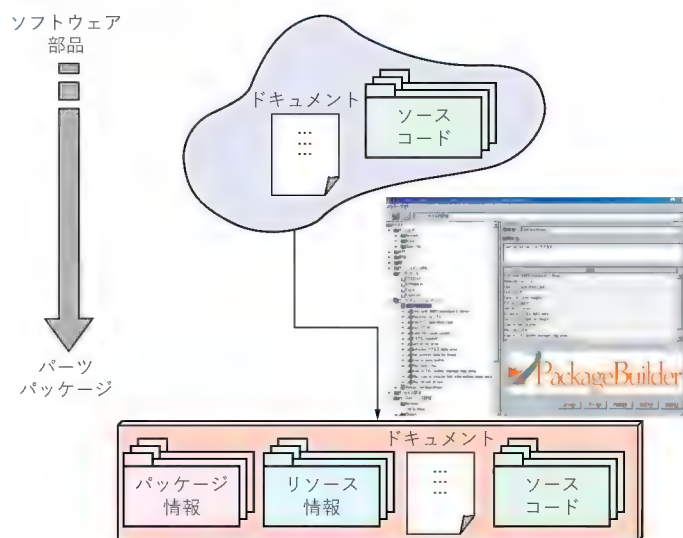
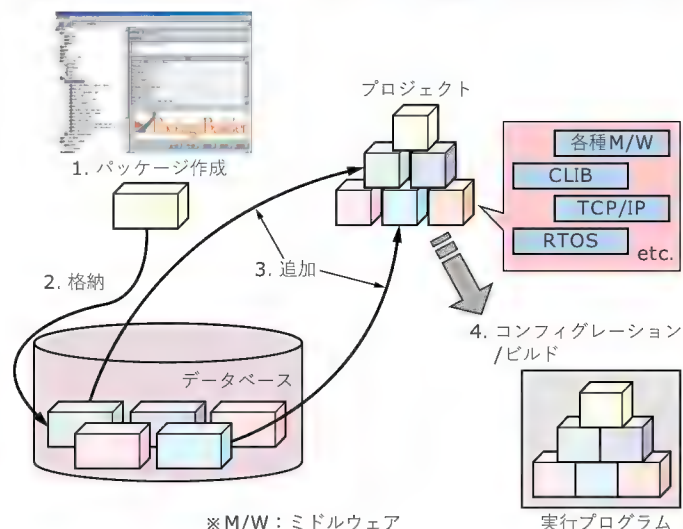
組み込みシステムは，機器ごとにハードウェアやソフトウェアの構成が異なるのが常です．使用するCPUやRTOSが変わるたびにターゲット環境をスクラッチから作成することは，非常に大きなコストを要します．そこで，ターゲット環境を共通化させることにより，使用できるものはそのまま使用したいという声が開発現場からしばしば聞かれます．それに応える環境として最近注目を浴びているのが，**T-Engine**(http://www.t-engine.org/)です(**図22**)．T-Engineは，ユビキタスコンピューティングを実現するためのターゲット環境を提供しており，ハードウェア，ソフトウェアの標準的な仕様を定めています．

T-Engineのハードウェア規格のコンセプトとして，

- 標準的なチップの採用
- 全回路・FPGAデータの公開
- 異なるCPUを載せた場合でもチップを極力共通化
- 統一化された拡張バスインターフェース

があります．このT-Engineボードを，ソフトウェア部品の再利用や，カスタムボード設計時のリファレンスとして使用できます．

また，RTOS環境の標準化も進めています．ソフトウェア部品の再利用促進をさせるためにレベル分けなどのサブセット化のための仕様は定めません．原則として，すべてのRTOSはす

〔図19〕パーツパッケージ

〔図20〕パーツパッケージのデータベースへの登録・プロジェクトへ追加

べての仕様を実装するよう規定されます．

T-Engineを使用することで，次のようなメリットを享受できると考えています．

① 標準部品の活用

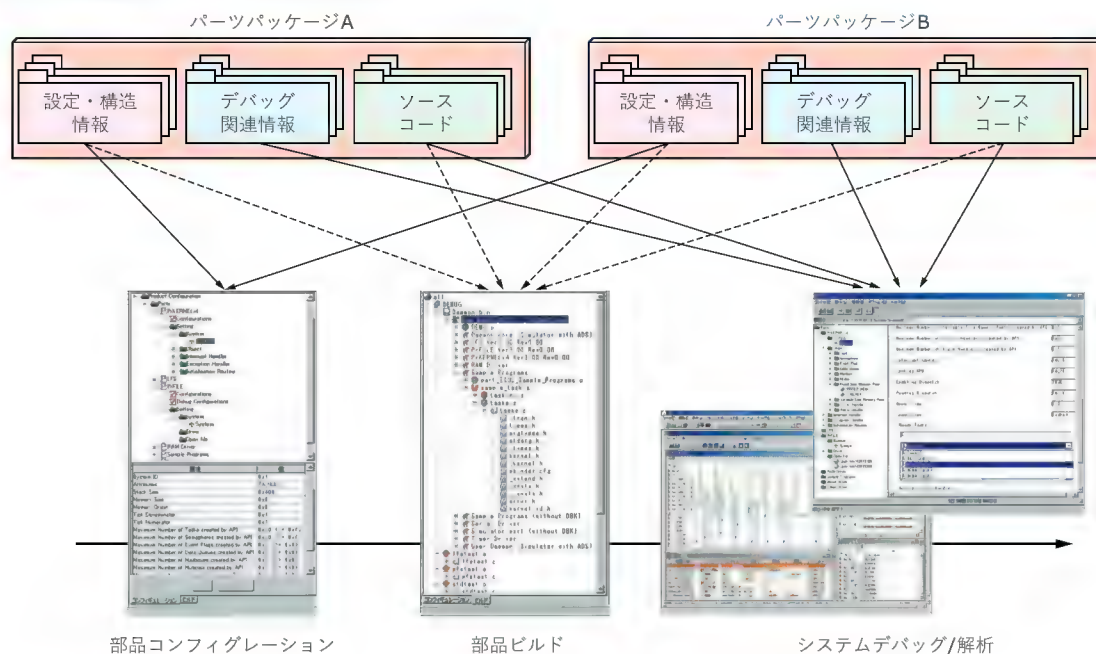
ハードウェアを共通化させ，その上で動作するさまざまなソフトウェアを標準部品化し，流通させることにより，組み込みシステム業界全体の開発効率の向上を図ります．

② オリジナル部品のフレームワーク

拡張ボードのインターフェースやハードウェア仕様を公開することで，カスタム部品を作るためのフレームワークを提供することができます．

③ 技術者の教育

ハードウェア，ソフトウェアの仕様をオープンアーキテクチャとして標準化されているため，技術者の教育環境として使

〔図21〕eBinderの各種機能によるリソース情報の利用

〔図22〕T-Engineボード

用できます.

しかし，まだT-Engineが登場して日も浅いため，世の中に浸透していくには，もう少し時間がかかるでしょう.

おわりに

組み込みシステムは，一にも二にもコストパフォーマンスが重要視されます．半導体技術の革新にともない，組み込みソフトウェアで高度な機能を実現する流れは今後も変わらないでしょう．組み込みソフトウェアが大規模化・複雑化していく一方で，開発環境が昔からさほど変わっていないことが問題だといえるでしょう.

この問題を解決するために，「次世代の組み込みシステム開発環境」に求められる要素について述べてきました．まとめると，次のようなことがいえます.

① 統一化された開発手法の提供
② 開発プラットホームの標準化に向けた環境整備
③ 技術者の教育を意識した機能要素の取り込み

組み込みシステムに携わる技術者として，忘れてはいけないことが一つあります．組み込みシステム開発の現場を支えているのは，開発ツールでもなく，ターゲット環境でもありません．現場の技術者です．次世代の開発環境に「技術者の教育」を求めるのは，そのためです．数年後には，組み込みシステム開発の問題点に「技術者の不足」がなくなることを期待します.

参考文献

1) 永井 正武ほか，『実用 組み込みOS構築技法』，共立出版
2) 上山伸幸，「Embedded System 設計手法の基礎知識」，『インタフェース』，1998年5月号
3) 斉藤光男，「半導体技術の進歩とシステムオンチップ」，『東芝レビュー』，Vol.57 No.1
4) 岩田茂人，「組み込みシステム向け開発環境に望まれているもの」，
5) (社)トロン協会，「ITRON仕様の最新動向」
(http://www.t-engine.org/ja/tu75/system.html)

みやはら・じゅんいち　イーソル(株)

豊富な採用実績を誇る
組み込み分野へのBSDの適用

本章では，組み込み機器への採用実績も多いといわれる，BSD系のOSに関する用語を解説する．本誌2002年8月号の特集「組み込み分野へのBSDの適用」では，BSDを組み込み分野へ導入するための基礎知識，ポーティングの実際，デバイスドライバの作成，カーネルのチューニング，そしてBSDを実際に導入した製品の事例までを徹底的に解説した．

BSD系OSは，長年の実績にもとづいた安定性，マルチプラットホーム対応，理解しやすい内部構造，洗練されたカーネルソースなどの面で評価が高い．とくに，NetBSDの移植性の高さには定評がある．また，商用利用を考慮したライセンス形態をとっている製品もあるなど，組み込み機器開発向けに適した特徴をいろいろもっている．

（編集部）

NetBSD移植の実際/NetBSDの組み込みシステム的チューニングとデバッグ

anonymous CVS

CVSの機能の一つで，匿名アカウントを用いて情報を参照する機能．この機能を使うことで，多人数で安全に情報の参照ができる．通常のCVSは，そのサーバにアカウントがないと使用できない．

CVS

ソフトウェアのバージョン管理システム．複数人でソースを共有し開発する場合に有効．実験用やリリース用などの枝分かれをさせて管理することが可能．

DDB

MachおよびBSD系OSに搭載されているカーネル内蔵のデバッガ．外部にデバッグ環境を用意しなくてよい．

diff形式

diffコマンドで出力された形式．複数の形式がある．patchコマンドを使ってdiffの出力を元ファイルに適用することが可能．

JNUG

日本NetBSDユーザーグループの略称(http://www.jp.netbsd.org/，**図1**)．国内のNetBSD関連のもっとも大きな団体．

KGDB

gdbを用いてカーネルのデバッグを行うシステム．カーネル側にgdbとの通信を行うスタブを用意する必要がある．

MI〔Machine Independent（マシン非依存）〕，MD〔Machine Dependent（マシン依存）〕

OS設計時に，マシンに依存する部分と非依存な部分を明確に分けることで，移植性，安定性を高めたり，ソースの肥大化を避けるなどの効果が期待できる．

NetBSD

BSD系UNIXのうちの一つ(http://www.netbsd.org/)．きれいな設計，高い移植性などの特徴をもつ．ロイヤリティフリーでかつすべてのソースコードが手に入る．

patch コマンド

diff形式のデータをもとに，ファイルに変更を加える．意図しない変更が加えられることもある．

sup

ソフトウェア（おもにソース）の自動更新を行うプロトコルおよびプログラム名．CVSを使うのが主流となり，supはあまり使われなくなった．

wasabi systems

NetBSDを商用サポートしている会社の一つ(http://www.wasabisystems.com/)．

カーネルトレース

多くのOSが，カーネル内部の動作情報をログとして残す機能を標準でもっている．おもにカーネルではなくユーザーランドのデバッグに用いられる．ユーザーランド側がカーネルに対してリクエストを発行し，どのような動作をしたのかがわかるので便利．

共有ライブラリ(shared library)

複数のプログラムでメモリ空間を共有し，そこにライブラリを置くことでメモリの利用効率を上げることができるライブラリ．

クラッシュダンプ

カーネルが復旧不可能状態に陥ったとき，デバッグ用としてメモリの内容をディスクに

〔図1〕JNUG(http://www.jp.netbsd.org/)

書き出す機構やそのデータを指す．

クロスコンパイル

ターゲットハードウェア上にあるコンパイラでソースのコンパイルを行うのでなく，異機種でコンパイル作業を行うこと．たいていの組み込み機器はハードウェア資源が限られているため，実機上にコンパイル環境を構築するのは現実的でなく，クロスコンパイルという形態がとられる．

チューニング

システムを適切に調節すること．具体例としては，システムの性能を上げるためにパラメータの変更を行うなど．

バグデータベースシステム

バグの内容，履歴などを管理するシステム．GNU gnatsが有名．

マルチプロセッサシステム

一つのシステムに複数のCPUをもつもの．高速なシステムを構築できるが，OSの設計および実装が難しい．

ユーザーランド(userland)

カーネル外部で動作するもの．コマンド，ライブラリなど，広範囲なものを指す．

NetBSDのデバイスドライバ開発

BUS SPACE機能

デバイスの中には，デバイスそのものへのアクセス方法は同じでも，接続されるバスの形態や種類が異なるバリエーションをもつものが多い(たとえば，同じシリアルデバイスを搭載するISAバスとPCIバスのカードが存在するなど)．このため，近年のUNIXでは，バスとデバイスの扱いを分離してバス機能を抽象化することにより，デバイスドライバを変更しなくても複数種類のバスで動作可能にしているものが多い．BUS SPACE機能は，NetBSDにおけるバス抽象化の実装で，デバイスドライバからバスへのアクセスを行うためのAPIをまとめたものである．

DMA転送(direct memory access)

デバイスとメモリ間のデータ転送を，プロセッサの命令実行によるデータのコピーをともなわずに自動的に行う機能で，プロセッサによるデータコピーより高速にデータを転送できる．システム上に，メモリとデバイス間のデータ転送を行う専用のDMAコントローラを設けて行う方式と，デバイスが自らメモリに直接アクセスする機能をもつバスマスタDMA方式がある．PCIバスがバスマスタDMA方式をサポートしていることから，近年ではバスマスタDMA方式が主流となっている．

soft state構造またはsoftc構造

デバイスドライバが複数のデバイスユニットを扱う際，各ユニットごとの状態などを保持するために使用するデータ構造．NetBSDではsoftc構造と呼ばれており，デバイスドライバとカーネル間の情報共有の機能も担っている．

UNIX System V

AT&Tで開発され，ライセンスされたUNIXのバージョンで，1983年に最初のリリースであるSystem V Release 1が発表された．1989年に発表された4番目のリリースであるSystem V Release 4をもって，UNIX System Vシリーズの開発は終了している．現在の多くの商用UNIXは，System V Release 4をベースにしたものであり，また多くのUNIX系OSがSystem Vの機能を採り入れており，現在のUNIX系OSの元祖といえるものである．

カーネルスレッド

カーネル内部で動作し続ける実行主体を得ることができる機能．通常，UNIXでは，カーネルは独立した実行主体をもたず，ユーザープロセスのサブルーチンのようにふるまうが，カーネルスレッド機能を用いると，ユーザープロセスとは関係なく特定の処理を行わせることができる．

仮想記憶

プロセスに対して，システムに搭載されている実メモリ容量を越えるメモリ空間が存在しているかのように見せかけるしくみで，近年のUNIX系OSのほとんどで実装されている．仮想記憶のもとでは，ユーザープロセスのメモリアドレスはカーネルのメモリアドレスと異なっており，またユーザープロセスの連続したメモリ空間が必ずしも物理的に連続していないこともあるため，デバイスドライバではアドレス変換やスキャッタ・ギャザDMAなどの特別な処理を行う必要がある．

キャラクタ型デバイス

UNIXにおけるデバイスの分類の一つで，シリアル端末などおもに文字単位でのアクセスを行うデバイスが該当する．本来はユーザーが操作するTTY端末を扱うためのものだったが，デバイスに対する直接I/Oを行うインターフェースも用意されているため，現在では汎用のデバイス型として扱われている．

コールアウト

伝統的なUNIXのカーネルにおいて，指定した時間後に指定した関数を実行させる機能．デバイスドライバによるタイムアウトの実現などによく利用される．

スキャッタ・ギャザDMA機能

デバイスとメモリ上のバッファでデータ転送を行う際，仮想記憶によりメモリ上のバッファが複数の断片に分割され，不連続な位置に配置されてしまう場合に，一度のDMA転送で自動的に複数の断片に正しい順序でデータを転送する機能．バスマスタDMA転送が可能なデバイスでサポートされていることがある．スキャッタ・ギャザDMA機能がサポートされていない場合，各断片ごとに個別にDMA転送を実行させることが必要となり，オーバヘッドが大きくなる．

スキャッタ・ギャザマップ機能

デバイスとメモリ上のバッファでデータ転送を行う際，仮想記憶によりメモリ上のバッファが複数の断片に分割され，不連続な位置に配置されてしまう場合に，これを連続した単一のバッファとしてバス上に割り当て，デバイスから透過的にバッファアクセスさせる機能．スキャッタ・ギャザマップ機能は，主記憶からバスへのアドレス変換機能であり，システム上に専用のハードウェアを必要とする．スキャッタ・ギャザマップ機能をもつシステムでは，デバイスでスキャッタ・ギャザDMA機能がサポートされていなくても，オーバヘッドなくDMA転送を行える．

静的構成(ドライバモジュール構成方法)

ドライバモジュールをカーネルの実行ファイルに組み込む方法の一つで，カーネル実行ファイルのコンパイル時に，ドライバモジュールも同時にコンパイルし，静的にリンクして実行ファイルを得る方法．
→動的構成

通常モード(ユーザーモード)

UNIX系OSでは，ユーザープロセスはプ

ロセッサの通常モード(ユーザーモード)で，カーネルはプロセッサの特権モード(カーネルモード)で実行される．通常モードでは，一部の特権命令は実行できず，アクセスできるメモリ空間にも制約が課せられるため，ユーザープログラムの誤りによりカーネルやシステムが破壊される危険を排除できる．

デバイス自動構成

カーネル起動時(またはデバイスドライバモジュールの動的構成時)に，デバイスユニットをスキャンし，存在するユニットのみにsoft state構造を割り当て，動的に初期化する機能．近年のほとんどのUNIX系OSで実装されている．デバイス自動構成機能により，たとえばディスクの増設など，同種のユニットの増減については，カーネルの再構築をともなわずに対応できる．

デバイスノード

デバイスユニットを表す論理的な識別子で，メジャー番号とマイナ番号を指定したmknod()システムコールにより，ファイルシステム上に作成される．デバイスノードは，ユーザープロセスからはファイルノードと同様に見え，通常の入出力システムコールを用いてアクセスできる．

→メジャー番号
→マイナ番号

動的構成(ドライバモジュール構成方法)

ドライバモジュールをカーネルの実行ファイルに組み込む方法の一つで，ドライバモジュールは個別にコンパイルしておき，カーネルの稼働中に必要なモジュールのみを動的に読み込み，リンクする方法．

→静的構成

特権モード(カーネルモード)

UNIX系OSでは，ユーザープロセスはプロセッサの通常モード(ユーザーモード)で，カーネルはプロセッサの特権モード(カーネルモード)で実行される．特権モードでは，特権命令を含むプロセッサのすべての命令と，すべてのメモリ空間にアクセスすることができる．

ドライバエントリポイント関数

UNIXカーネルに対するデバイスドライバのAPI関数．デバイス自動構成機能，標準入出力機能，デバイス特殊制御機能の3種類の関数がある．詳細はOSごとに異なるが，NetBSDでは次のように定義されている．

▶デバイス自動構成機能
- match()**関数**
  デバイスユニットの存在確認
- attach()**関数**
  デバイスユニットの初期化
- detach()**関数**
  デバイスユニットの切り離し
- activate()**関数**
  デバイスユニットの有効化/無効化

▶標準入出力機能
- open()**関数**
  デバイスユニットのオープン処理
- close()**関数**
  デバイスユニットのクローズ処理
- read()**関数**
  デバイスユニットからのストリーム入力
- write()**関数**
  デバイスユニットへのストリーム出力
- strategy()**関数**
  デバイスユニットへのブロック入出力

▶デバイス特殊制御機能
- ioctl()**関数**
  デバイスユニット特有の制御処理
- poll()**関数**
  デバイスユニットの非同期待ち合わせ処理
- mmap()**関数**
  デバイスユニットのメモリ空間マップ処理

注1：match()関数は，NetBSD以外のUNIX系OSでは，probe()という名称であることが多い．

注2：厳密にはドライバエントリポイント関数ではないが，割り込みハンドラもエントリポイント関数に含めて論じられることがある．

ブロック型デバイス

UNIXにおけるデバイスの分類の一つで，ディスク装置など，おもにブロック単位でアクセスを行うデバイスが該当する．ブロック型デバイスに対しては，メディア上にファイルシステムを構築することができ，またカーネルによるブロックバッファリングを介してアクセスされる．

マイナ番号

デバイスノードからデバイスユニットを特定するために使用される，デバイスドライバ依存の識別番号．通常は，デバイスのユニット番号をそのままマイナ番号として使用するが，マイナ番号の解釈は完全にデバイスドライバにまかされているため，アクセスのモードやパーティションの番号など，ユニット番号以外の意味をもたせることもある．一般的には，マイナ番号を複数のビットフィールドに分割し，そのうち一つをユニット番号に割り当て，他のフィールドをそれぞれ個別の機能に割り当てることが多い．

→デバイスノード
→メジャー番号

メジャー番号

デバイスノードからデバイスドライバモジュールを特定するために使用される，デバイスドライバモジュールの識別番号．デバイスノードにはメジャー番号とマイナ番号が記録されており，デバイスノードに対するアクセスはカーネルによって同じメジャー番号をもつデバイスドライバモジュールのエントリポイント関数の呼び出しに変換される．

→デバイスノード
→マイナ番号

ユニット番号

単一のデバイスドライバが扱う複数のデバイスユニットに対して，それぞれ一意に与えられる識別番号．デバイスドライバの各エントリポイント関数には，操作対象となるデバイスユニットのユニット番号が引き数として渡される．エントリポイント関数では，ユニット番号より当該ユニットの状態を保持するsoft state構造を取り出し，そこに含まれる情報にしたがってデバイスの制御を行う．

FreeBSD環境の構築

chobitBSD

広島大学の山岡氏が開発した，フロッピディスク2枚構成で使えるファイアウォールディストリビューション．現在は大学を卒業されたため，関連サイトは閉鎖中．picoBSDを参考にしている．

Floppy-1プロジェクト

誰もが簡単に，素早く，使えるファイアウォール兼VPNゲートウェイ専用のディストリビューションが欲しいという開発者の欲求から開始されたプロジェクト．配布サーバと連携するのが特徴．

FreeBSD

Bill Jolitzの開発した386BSDのパッチキットの開発から発展したPC/AT互換機用のフリーなUNIX．当初はi386アーキテクチャだけだったが，最近は対応CPUが増加してい

る．表1にBSDの特徴とパッケージ管理システムをまとめる．

Kameプロジェクト

次世代プロトコルのIPv6を，フリーに＊BSDで利用できるようなIPv6/IPsecスタックを開発しているWIDEプロジェクトの一つ．IPv6/IPsecのリファレンスコードとして利用されている．

LRP（Linux Router Project）

Linuxでもっとも有名な，フロッピーディスク版ルータを開発するプロジェクト．これから派生したフロッピーディスク版ディストリビューションは多数存在する．サイトは
http://www.linuxrouter.org/

NetBSD

386BSDのパッチキットの開発から発展したフリーなUNIX．当初からマルチCPU，マルチアーキテクチャを意識して開発されていたため，組み込みOSでの利用がさかんであり，中心開発者には日本人が多い．**表1**にBSDの特徴とパッケージ管理システムをまとめた．

OpenBSD

386BSDのパッチキットの開発から発展したフリーUNIX．セキュリティを重視した開発スタイルをとっている．OpenSSHの開発もこのプロジェクトから派生している．最近ひそかに注目株である．**表1**にBSDの特徴とパッケージ管理システムをまとめた．

picoBSD

FreeBSDのパッケージに含まれているフロッピディスク版ディストリビューション開発キット．ルータ版，ブリッジ版など数種類存在する．Floppy-1プロジェクトでも参考にしている．

Plan9

UNIXを開発したBell研究所によって，1995年にリリースされた新しいOS．特筆すべき点は，ネットワークの利用を前提に開発されていること．現在は学術的・趣味的な利用が大半だが，次世代OSの代表格になるかもしれない．

PPTPサーバ機能

マイクロソフトのWindows NT4.0から採用されたクライアント-サーバ型のVPN方式/プロトコル．Windows 98以降，VPNクライアントは標準搭載されているため，比較的よく利用されている．

vnodeディスク

ファイルとして存在しているディスクイメージを仮想的なディスクとして扱う疑似ディスクデバイス．フラッシュメモリに焼き込むためのディスクイメージを作成するのに利用したりする．

クランチバイナリ

複数のコマンドを一つの実行ファイルにまとめたバイナリファイル．各コマンドのコンパイル時に作成されたオブジェクトファイルと参照されるライブラリをすべて一つの実行ファイルとしてリンクして作成する．

スケルトン

ソースをコンパイル・インストールするタイプのパッケージ管理システム（portsやpkgsrc）において，そのマシン上でコンパイルしインストールするために必要となる最小限のファイルのセットのこと．

パッケージ

プログラムの配布をコンパイルしたバイナリと付属のヘルプファイルなどを一つのアーカイブにまとめて配布したもの．LinuxでのRPMやdeb，BSD系のportsやpkgsrcが有名．

パッケージ管理システム

RPMやdeb，portやpkgsrcなどのパッケージ全体とパッケージの管理システム．これを利用することで，一つのコマンドと複数の引き数でプログラムのインストールやアンインストールが可能となる．

mmEyeと電源即断環境に対応したファイルシステム

BSC（Bus State Controller）レジスタ

日立製のCPUであるSHシリーズのバス幅，メモリアクセス速度，リフレッシュレートなどCPU外部バスの基本的な機能を設定するレジスタ群のこと．

FFS（Fast File System）

NetBSD/FreeBSD/OpenBSDなどのBSD系のOSで採用されている標準のファイルシステムの名称．

FIFOなしのSCIとFIFO付きのSCIF

日立製のCPUであるSHシリーズに搭載されている，非同期シリアル通信を行うコントローラで，16バイトの送受信バッファ用FIFOを内蔵しているSCIFと，内蔵していないSCIとがある．

〔表1〕BSDの特徴とパッケージ管理システム

	特　徴	対応アーキテクチャ	パッケージ管理システム	パッケージ数（アプリケーションソフト数）
FreeBSD	インテルi386を中心に開発されており，利用者はもっとも多い．ユーザーにとって利用しやすいOSをめざして開発している．NetBSD，OpenBSDの元になる	alpha，ia64，ppc，sparc64，x86_64	ports	6845（2002.6時点）
NetBSD	i386系以外の他のCPUへの移植をめざして開発されており，対象CPUは最多．カーネルはCPUアーキテクチャ依存と非依存の部分で開発されており，移植性を大事にしている	alpha，arm，arm32，hppa，i386，m68k，mipseb，mipsel，ns32k，powerpc，sh3eb，sh3el，sparc，sparc64，vax，x86_64	pkgsrc	2783（2002.6時点）
OpenBSD	暗号機能の統合などセキュリティを重視した開発スタイルをとっており，OpenSSHなど他のOSでも使用されているセキュリティ関連ソフトを多数開発	alpha，amiga，hp300，i386，mac68k，macppc，sparc，sparc64，sun3，vax	ports	1871（2002.6時点）

JPEGエンコーダ

圧縮されていない画像データを圧縮形式の一つであるJPEG形式に変換するLSIもしくは，変換を行うプログラム．

tarファイル

名称の語源はTape ARchiverの略で，磁気テープに複数のファイルを格納する目的のために開発されたものだが，現在はディスク上の複数のファイルを一つのファイルにまとめるために使用されることのほうが多い．

rawデバイス

BSD系UNIXにおけるデバイスドライバの一種で，ディスクドライバなどのブロックデバイスに対し，バッファキャッシュを通さずにアクセスする手段を提供するもの．

永続性

あるデータが永続性をもつとは，電源断やシステムリセット後もそのデータが同一の値を保持できることである．ハードディスクに格納されているデータは通常永続性があり，メモリ内にだけ存在するデータは通常永続性がない．

疑似ドライバ

UNIXにおけるデバイスドライバの一種で，実際のデバイスを制御するためのものではなく，特定の機能を提供するために存在するもの．たとえば，メモリディスクやネットワークパケットフィルタなどがある．

特権モードプロセス

通常のプロセスでは特権エラーとなる物理アドレスの参照や，I/Oポートを直接アクセスできるような権限をもったプロセス．

ブートシーケンス

コンピュータの電源を入れてから，目的とするプログラムが動作するまでの一連の動作のこと．通常はCPUのレジスタ/周辺デバイス/メモリの初期化，OSの起動，アプリケーションの実行などが行われる．

RTLinuxのリアルタイム拡張手法とRTCore/BSD

mutex

スレッドプログラミングで排他制御を提供する機構の一つ．

〔図2〕
OSの中におけるRTLinuxの位置付け

RTLinux

FSMLabs社が開発したLinuxのハードリアルタイム拡張機能．GPLで配布されているオープンソース版と，非GPLの商用版(RTLinux/Pro)がある．OSの中におけるRTLinuxの位置付けを**図2**に示した．

RTCore/BSD

FSMLabs社によるNetBSDのハードリアルタイム機能拡張．

RTL-FIFO

RTLinuxが提供する通信機能．カーネル空間で動作するリアルタイムモジュールとユーザー空間で動作するアプリケーション間での相互通信を実現する．

スレッド

プログラミングにおけるスレッドとは，プロセスの中におけるプログラムの実行単位．スレッドはプロセス中に複数存在することができ，プロセスの資源を共有する．

ハードウェア割り込み

周辺デバイスがCPUに対して何らかの要求を出すときに発生する割り込み．

排他制御

複数のスレッドがプロセスの資源にアクセスする際，同時にアクセスが発生するとデータ破壊が起こる．この同時アクセスを抑制するための機構を，排他制御と呼ぶ．

ハイブリッドアーキテクチャ

異なるアーキテクチャをもつものが協調しあって動作するしくみ．

プリエンプティブカーネル

マルチタスクOSはハードウェアタイマを使ってアプリケーションを強制的に切り替えることによって各プロセスに対して均等にCPUを割り当てる．この考えをカーネル内部にも適応したもの．

齊藤正伸　(株)インターネットイニシアティブ
遠藤知宏　(株)創夢
西山英之　フロッピーワン(有)
堀内岳人　(株)ブレインズ
渡辺淳一　(株)エーアイコーポレーション

Information ── JPEG2000対応ICがNHKのハイビジョンリアルタイム圧縮伸張基板の開発に貢献
アナログ・デバイセズ社は，同社のJPEG2000対応ICチップ「ADV202」が，NHK放送技術研究所による世界初のパソコン用ハイビジョンリアルタイム圧縮伸張基板の開発に貢献したことを発表した．

chapter 03

コンピュータにより可能になった新たな科学/工学分野

基礎からの計算科学・工学 ——シミュレーション

本章では，コンピュータで行うさまざまなシミュレーションに関する用語を解説する．コンピュータとは，いうなれば「すべてを模式的に実現する装置」であり，コンピュータの用途としてのシミュレーションは，あらゆる分野において重要な役割を果たしている．実際に物を作り，測定器をつながなくても結果が見える，観測が難しいものでも結果を予測することができるなど，工学・科学分野に多大な利益をもたらしている．2002年9月号の特集「基礎からの計算科学・工学——シミュレーション」では，物理シミュレーション，回路シミュレーション，制御シミュレーション，離散系シミュレーションなどについて，考え方や実際の手法を解説した．

シミュレーションで重要なのは，対象のモデリングである．そして正確なモデリングとは，対象の理解そのものでもある．シミュレーションは，使い方を間違えると当然間違った結果しかもたらさないので，細心の注意が必要となる．

(編集部)

世界を僕のマシンの上に/コンピュータシミュレーションをしよう

Boids

鳥や魚が群れになって移動するようすをコンピュータ上に再現するためにCraig Raynolds提案したモデル．各個体はいくつかの簡単な行動規則にしたがって移動するだけで，リーダーもいないのだが，全体としていかにも群れらしい運動が見られる．

GRAPE

東京大学で開発された重力計算専用コンピュータ．多くの天体が重力を及ぼしあいながら運動する状況をシミュレーションしようとすると，各天体が他のすべての天体から受ける力の総和を計算する部分にもっとも多くの計算時間を要する．GRAPEはその重力の計算に特化した計算機であり，浮動小数点演算速度で世界最高の63TFLOPS（1秒間に63兆回の演算を行う）を達成している．

カオス現象

$f(x) = 4 \times (1 - x)$

という関数（「ロジスティック写像」と呼ばれる）のxとして0と1の間の適当な数値を代入し，出てきた結果を再びxに代入するという操作を繰り返すと，得られる数列は**図1**のように一見不規則に激しく上下する．このように，決まった操作から不規則なふるまいが出現することを「カオス」と呼ぶ．カオス現象は，3個以上の物体が力を及ぼし合いながら運動するような場合やある種の電気回路など，さまざまな状況で出現する．

〔図1〕
ロジスティック写像から発生するカオス

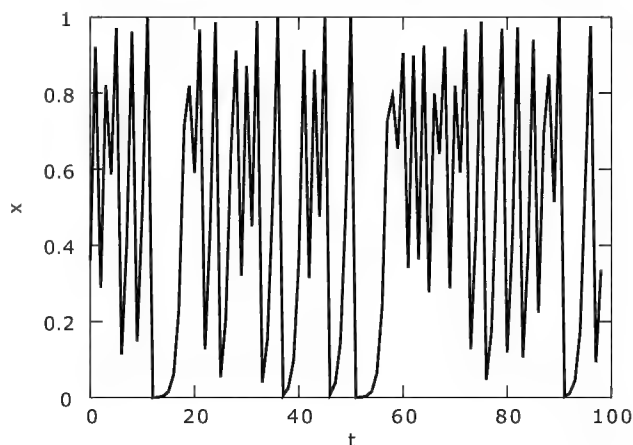
拡散方程式

水に落としたインクが拡がっていく場合のように，ブラウン運動する粒子が大量にあるとき，その密度が場所ごとにどのように時間変化していくかを記述する簡単な偏微分方程式．また，熱の伝導も同じ形の偏微分方程式で記述される．

基礎方程式

力学の問題ならニュートンの運動方程式，電磁気ならマックスウェル方程式というように，さまざまな物理現象にはそれを記述するための基礎となる方程式がある．もっとも，「基礎方程式」という言葉の使い方はもう少しあいまいで，個別の状況までを考慮に入れたものを基礎方程式と呼ぶ場合もあるようだ．

シミュレーション

もともとは「模倣する」ことを意味する英語だが，日本では模疑実験や模疑演習などにほぼ限定して使われるようだ．実物では実験できない状況や製品設計段階で実物が存在しない場合などに，実物のふるまいを予測・検証するためにシミュレーションが行われる．最近では，数理的なモデルにもとづくコンピュータシミュレーションが主流になってきており，それを単に「シミュレーション」と称することも多い．分子のミクロな運動から，

地球大気全体をモデル化して気候の変動を調べる大気大循環モデルまで，科学技術のさまざまな分野でコンピュータシミュレーションが使われている．

創発現象

Boidsでは個体の行動規則だけを設定することにより，群全体としての運動が出現する．このように，簡単な規則にしたがって動く要素を多数集めた結果，全体として思いもかけないふるまいが現れることを「創発」と呼び，複雑系科学の基本概念となっている．もっとも，思いがけないことかどうかの判定には主観的な解釈がともなう．

地球シミュレータ

気象や地殻変動など地球規模の現象を精度よくシミュレーションすることを目的に作られた日本のスーパーコンピュータシステム．5000台を越えるスーパーコンピュータを高速ネットワークでつないだ並列計算機であり，重力専用計算機GRAPEには及ばないものの，汎用のスーパーコンピュータシステムとしては世界最高の演算速度を誇る．

チューリングマシン

イギリスの数学者アラン・チューリングが「計算」の理論を構築するために提唱した仮想的な機械で，無限に長いテープとその上を動くヘッドによって構成される．テープには記号が書き込まれ，ヘッドはその記号を読みながらそれに応じて内部の状態を変えたりテープの記号を書き換えたりという動作を行う．理論的には，「計算」とはチューリングマシンによって有限のステップ数で実行できる記号操作のことと定義される．現代のコンピュータは，この仮想的なチューリングマシンを実現しようとしたものと考えることができる．

ナビエ・ストークス方程式

流体力学の基礎となる偏微分方程式．流体のふるまい，たとえば水の流れや渦の生成・消滅，対流，乱流などはすべてこの方程式によって記述される．流体のシミュレーションとは，結局この方程式をコンピュータで解くことにほかならない．

分子動力学

液体などを分子1個1個が見える程度のミクロなレベルでシミュレーションするための方法の一つ．実際には，分子間に働く力を計算し，ニュートン力学の運動方程式を解いて分子を動かしていくだけである．しかし，たとえば水を冷やして氷にするというただそれだけのシミュレーションですら非常に難しく，最近になって名古屋大学のグループが世界で初めて水を凍らせる分子動力学シミュレーションに成功した．

偏微分方程式

拡散方程式は，ある量を時間で偏微分したものが同じ量を座標で偏微分したものと関係するという形式で書かれている．このように二つ以上の変数に関する微分を含む微分方程式を総称して「偏微分方程式」と呼ぶ．

マックスウェル方程式

静電磁場であれ電磁波の放射であれ，電磁気現象は一連の偏微分方程式の組によって記述される．この方程式群をまとめて「マックスウェル方程式」と呼ぶ．電磁場や電磁波の解析とは，与えられた条件のもとでこの方程式を解くことにほかならない．

メルセンヌ・ツイスター

松本眞・西村拓士両氏によって提案された乱数生成法．計算によって生成される乱数（本当は乱数ではないので「疑似乱数」と呼ばれる）は，必ず周期をもつ．メルセンヌ・ツイスターは周期が非常に長く，モンテカルロ法のための質の良い乱数と考えられている．

モンテカルロシミュレーション

放射線元素の崩壊やブラウン運動など確率の要素が含まれる現象を乱数を用いてシミュレーションする方法の総称．「熱」の効果を扱う場合にもよく用いられる．カジノで有名な町の名前にちなんでこの名がつけられた．

ルンゲクッタ法

常微分方程式を解くための代表的な数値計算アルゴリズム．コンピュータは微分を扱えないので微分は差分で近似することになるが，当然近似の精度は高いほうが良い．ルンゲクッタ法はその近似の精度を上げる手法の一つで，近似の精度が異なる一連の方法の総称である．

連立常微分方程式

一つの変数に関する微分だけを含む微分方程式が常微分方程式である．たとえばニュートンの運動方程式は，位置や速度の時間に関する微分だけを含む．何個かの常微分方程式の組み合わせによって一つの現象が記述されるような場合，「連立常微分方程式」と呼ばれる．

物理シミュレーションの手法と結果の検証

Mathematica

数式処理，数値計算，可視化などのさまざまな機能をもったプログラムパッケージ．Wolfram Research社の商品．類似のものとしてMaple，MATLAB（いずれも有償），Octave（無償）などがある．

オイラー法

常微分方程式を数値的に解く方法のうちもっとも簡単なものの一つ．常微分方程式，

$$\frac{dx}{dt}=f(x,t)$$

の時刻t_iでの近似解がx_iだったときに，時刻$t_{i+1}=t_i+\Delta t$での近似解x_{i+1}を，

$$x_{i+1}=x_i+\Delta t f(x_i,\ t_i)$$

として計算する方法．

常微分方程式の厳密解

誤差を含まない「正しい」解のこと．方程式によっては，三角関数，指数関数などで厳密解を書き下すことができる．厳密解が知られている常微分方程式の例には，線型常微分方程式，ケプラー問題（重力で相互作用する2体の運動）などがある．

シンプレクティック法

ハミルトン系を表す常微分方程式を数値積分する方法のうち，計算法自体が正準変換の性質をもつもの．直観的には，正準変換は方程式の性質を変えないので，シンプレクティックでない方法よりも良い性質をもつことが期待できる．実際に，系のエネルギが長時間積分してもずれていかないなどの良い性質をもつ．

台形則

常微分方程式，

$$\frac{dx}{dt}=f(x,t)$$

の近似解を，

$$x_{i+1}=x_i+\frac{\Delta t}{2}\left[f(x_i,t_i)+f(x_{i+1},t_{i+1})\right]$$

として計算する方法．

調和振動

単振動ともいう．$\ddot{x}=-kx$の形の2階定係数線型常微分方程式で記述される運動．解は

時間の三角関数 $x = C \sin \omega t$ (C, ω は定数)で与えられる．理想的な振り子や，バネにつながったおもりの運動は調和振動になる．また，一般に弾性振動は調和振動になる．

ハミルトン系

ハミルトン力学系ともいう．要するにニュートンの運動方程式によって記述される運動のことだが，摩擦や粘性などがない理想的な場合をさす．このとき，運動方程式をハミルトン形式（あるいは正準形式）に書くことができる．正準形式の運動方程式は，座標変換が正準変換の性質をもっていると，変換後も同じ形になる．

非線型振動

調和振動でない振動現象一般をさす．多くの自然現象や工学的な対象になる振動現象では，振幅が小さい場合には近似的に調和振動とみなせるが，振幅が大きくなるにしたがって非線型性が無視できなくなってくる．

リープフロッグ公式

シンプレクティック法のうちもっとも簡単なものの一つ．運動方程式が，

$$\frac{d^2x}{dt^2} = a(x)$$

であるとき，速度を v として，

$$v_{i+1/2} = v_{i+1/2} + \Delta t a(x_i)$$
$$x_{i+1} = x_i + \Delta t v_{i+1/2}$$

という形で，速度と位置を互い違いに進める方法．

回路シミュレーションの実際

AC 解析（交流解析）

回路シミュレーションにおける解析項目の一つ．DC 解析で求めた直流動作点を用いて，アナログ回路への小信号入力に対する周波数応答を解析すること．

DC 解析（直流解析）

回路シミュレーションにおける解析項目の一つ．アナログ回路の定常状態における電流値や電圧値を解析すること．AC 解析における直流動作点を求めるために行われる．

Ebers-Moll モデル

バイポーラトランジスタのデバイスモデルの一つ．このデバイスモデルは単純であり，トランジスタの物理特性をよく表しているため，SPICE などによる回路シミュレーションにおいてよく用いられる．

Frohman-Bentchkowsky モデル

MOS トランジスタのデバイスモデルの一つ．このデバイスモデルは 2 次元解析によって得られたモデルであり，Shichman-Hodges モデルに比べて，より厳密で複雑になっている．

Gummel-Poon モデル

バイポーラトランジスタのデバイスモデルの一つ．このデバイスモデルは Ebers-Moll モデルを改良したものであり，より厳密な回路シミュレーションをする場合によく用いられる．

LU 分解法

連立方程式の行列求解法の一つ．回路シミュレーションにおける AC 解析でよく用いられる．

Shichman-Hodges モデル

MOS トランジスタのデバイスモデルの一つ．このデバイスモデルは 1 次元解析によって得られた簡単なモデルであり，SPICE などによる回路シミュレーションにおいてよく用いられる．

アナログ回路

図 2(a) に示すような連続的な値をとる信号（これをアナログ信号という）を取り扱う電子回路であり，トランジスタ，抵抗，キャパシタなどのアナログ素子で構成される．アナログ電子回路ともいう．

回路検証

回路設計が正しく行われたかどうかを確認する作業の総称（**図 3**）．回路検証として，通常は，回路シミュレーションが行われる．

回路シミュレーション

回路検証の手段の一つ（**図 3**）．通常は，回路設計で作成した回路図と各アナログ素子の特性をもとに，コンピュータを用いた数値計算によって，回路のアナログ的な動作が解析される．

回路設計

LSI 設計における四番目の設計段階のこと（**図 3**）．この段階では，論理設計で作成した回路図（ネットリスト）から，トランジスタ，抵抗，キャパシタなどのアナログ素子のみで表した回路図（ネットリスト）を作成する．

過渡解析

回路シミュレーションにおける解析項目の一つ．指定した時間領域での電圧・電流の過渡的な応答波形を解析すること．

慣性遅延モデル

各ゲート回路が入力信号に応答するために必要な最小の時間を割り当てる遅延モデル．実際の回路でも，非常に短いパルスに対しては応答しないため，このモデルは比較的よく用いられる．

機能検証

機能設計が正しく行われたかどうかを確認する作業の総称（**図 3**）．

機能シミュレーション

機能検証の手段の一つ（**図 3**）．通常は，機能設計で決定した構成要素および構成要素間のデータの流れをハードウェア記述言語などで記述し，その動作を確認することによって行われる．

機能設計

LSI 設計における 2 番目の設計段階のこと（**図 3**）．この段階では，方式設計で決定したアルゴリズムやアーキテクチャを実現するための構成要素を決め，各構成要素間のデータの流れを表すブロック図を作成する．

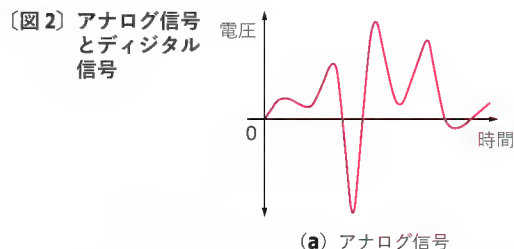
後退公式

数値積分で用いられる公式の一つ．後退公式を使用した数値積分法は，安定性が良いため，回路シミュレーションの過渡解析でよく

〔図 2〕アナログ信号とディジタル信号

(a) アナログ信号

電圧
1
0
時間

(b) ディジタル信号

用いられる．

最大最小遅延モデル

ゲート回路の種類（NOT ゲート，AND ゲート，OR ゲートなど）ごとに，遅延の最大値と最小値をもたせる遅延モデル．高精度のシミュレーションを行う際などに用いられるが，シミュレーション時間は長くなる．

ゼロ遅延モデル

各ゲート回路が遅延をもたないものとした遅延モデル．高速なシミュレーションを行えるが，精度が低いため，あまり用いられない．

タイムホイール

論理シミュレーションにおけるシミュレーション時刻と励起イベントを管理するためのデータ配列（テーブル）．配列が環状になっているため，タイムホイールと呼ばれる．

〔図 3〕LSI の設計過程

立ち上がり/立ち下がり遅延モデル

信号の立ち上がり（0 から 1 への変化）時と立ち下がり（1 から 0 への変化）時とで，異なる遅延をもたせる遅延モデル．高精度のシミュレーションを行う際などに用いられるが，シミュレーション時間は長くなる．

単位遅延モデル

すべてのゲート回路は同一の遅延をもっているものとした遅延モデル．高速なシミュレーションを行えるが，精度が低いため，あまり用いられない．

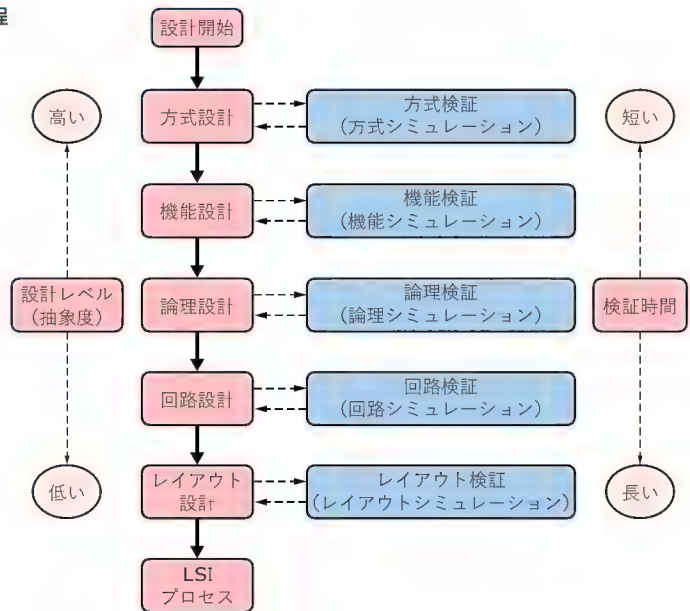
遅延モデル

論理シミュレーションにおいて，実際のディジタル回路の素子遅延時間を表現する方法．遅延モデルには，ゼロ遅延モデル，単位遅延モデルなどがある．

ディジタル回路

図 2（b）に示したような 0，1 などの離散的な値をとる信号（これをディジタル信号という）を取り扱う電子回路であり，NOT ゲート，AND ゲート，フリップフロップなどのディジタル素子で構成される．ディジタル電子回路またはロジック回路ともいう．

デバイスモデル

回路シミュレーションにおいて，実際のトランジスタなどのアナログ素子の素子特性を表現する方法．デバイスモデルには，Ebers-Moll モデル，Gummel-Poon モデルなど多数のモデルがある．

ニュートン・ラフソン法

非線形関数の代表的な数値計算法の一つであり，ニュートン法とも呼ばれる．回路シミュレーションにおける DC 解析でよく用いられる．

ネットリスト

回路図の表現形式の一つであり，電子回路の回路素子（アナログ回路ではトランジスタ，抵抗，キャパシタなど，ディジタル回路では NOT ゲート，AND ゲート，フリップフロップなど）の接続関係を表した設計データ．

ハードウェア記述言語（HDL）

ハードウェアの動作や構造を記述するための形式言語の総称であり，HDL（Hardware Description Language）ともいう．機能シミュレーションや論理合成などに用いられる．

標準遅延モデル

ゲート回路の種類（NOT ゲート，AND ゲート，OR ゲートなど）ごとに，異なる遅延をもたせた遅延モデル．実際の回路動作に比較的近いシミュレーション結果が得られるため，もっとも一般的に使用されている．

方式検証

方式設計が正しく行われたかどうかを確認する作業の総称（図 3）．方式検証として，通常は，方式シミュレーションが行われる．

方式シミュレーション

方式検証の手段の一つ（図 3）．通常は，方式設計で決定したアルゴリズムやアーキテクチャを，C/C++ などのプログラミング言語やハードウェア記述言語で記述し，それらの動作を確認することによって行われる．

方式設計

LSI 設計における最初の設計段階のこと（図 3）．この段階では，LSI の動作やその動作を実現するアルゴリズムやアーキテクチャなどの仕様を決定する．

ムーアの法則

米 Intel 社の創設者の一人であるゴードン・ムーア（Gordon E. Moore）が 1965 年に提唱した，「半導体の集積度は，1 年半ごとに約 2 倍になる」という経験則．

レイアウト検証

レイアウト設計が正しく行われたかどうかを確認する作業の総称（図 3）．レイアウト検証として，通常は，レイアウトシミュレーションが行われる．

レイアウトシミュレーション

レイアウト検証の手段の一つ（図 3）．レイアウト設計で作成したマスクパターンをもとに，信号線の幅や信号線の間隔が規定値を満足しているか，配線の短絡や切断はないかなどがチェックされる．

レイアウト設計

LSI設計における最後の設計段階(**図3**)であり，これより後は，製造工程(LSIプロセス)となる．この段階では，回路設計で作成した回路図をもとに，製造工程におけるフォトマスク原画となるマスクパターンを作成する．

論理検証

論理設計が正しく行われたかどうかを確認する作業の総称(**図3**)．論理検証として，通常は，論理シミュレーションが行われる．

論理合成

機能設計段階のHDL記述を論理設計段階のHDL記述や回路図(ネットリスト)に自動変換する技術．

論理シミュレーション

論理検証の手段の一つ(**図3**)．通常は，機能シミュレーションと同じように，論理設計で作成した回路図をハードウェア記述言語などで記述し，その動作を確認することによって行われる．

論理設計

LSI設計における3番目の設計段階のこと(**図3**)．この段階では，機能設計で作成したブロック図から，NOTゲート，ANDゲート，フリップフロップなどのディジタル素子のみで表した回路図(ネットリスト)を作成する．

信号処理におけるモデリングの例

Durbinのアルゴリズム(Durbin 's algorithm)

線形予測係数を求める場合に，いくつかの方法がある中でよく使われるのが，**図4**の連立方程式を解く方法である．

この式で，$r[j]$は相関関数と呼ばれ，線形予測法の対象となる信号を$x[n]$とすると，次のように定義される．

$$r[j]=\sum_{n=0}^{N-1-j}x[n]x[n-j]$$

このとき，左辺の$M\times M$の行列は，主対角要素(左上から右下への対角線上に並んだ要素)はすべて同じ値($r[0]$)になっている．また，主対角要素と平行する要素もそれぞれ同じ値になっている．さらに，この行列は対称行列である．このような場合に，この連立方程式を効率よく解く方法の一つがDurbinのアルゴリズムである．

IIRフィルタ(IIR filter)

IIRフィルタとは，入力信号が0になっても，出力信号が無限に0にならないようなフィルタである．IIRフィルタをディジタルフィルタとして実現する場合，その入力を$x[n]$，出力を$y[n]$で表すと，その両者の関係は次の差分方程式により表すことができる．

$$y[n]=a_1y[n-1]+a_2y[n-2]+\dots+a_My[n-M]+b_0x[n]+b_1x[n-1]+b_2x[n-2]+\dots+b_Kx[n-K]$$

これに対して，入力信号が0になると，ある時間の経過の後，出力信号が0になるフィルタをFIRフィルタと呼ぶ．

PARCOR係数(PARCOR coefficient)

標本化された離散的な信号において，2点間の相関係数を計算する際に，2点の間にある信号による影響を除いてから計算した相関関数は，統計学の分野では偏相関係数と呼ばれている．これがPARCOR係数である．ところで，偏相関係数に対応する英語はpartial correlation coefficientである．PARCORという語は，partial correlationの下線部から作られた造語で，最初に線形予測法の研究を行っていた日本人研究者が名づけたものである．

移動平均モデル(moving average model)

MAモデルとも呼ばれる．時系列信号において，ある線形システムの現在の入力信号$g[n]$および過去の入力信号$g[n-1]$，$g[n-2]$，……，$g[n-K]$の重み付きの和で，現在における信号$x[n]$を表すというモデルである．つまり，移動平均モデルでは，信号$x[n]$を次のような式で表現する．

$$x[n]=b_0g[n]+b_1g[n-1]+b_2g[n-2]+\dots+b_Kg[n-K]$$

自己回帰移動平均モデル(autoregressive moving average model)

自己回帰モデルと移動平均モデルを合わせたもので，ARMAモデルとも呼ばれる．このモデルでは，現在における信号$x[n]$を，現在の入力信号$g[n]$，過去の入力信号$g[n-1]$，$g[n-2]$，……，$g[n-K]$，過去の出力信号$x[n-1]$，$x[n-2]$，……，$x[n-M]$の，重み付きの和で表す．つまり，自己回帰移動平均モデルでは，現在における信号$x[n]$を次のような式で表現する．

$$x[n]=a_1x[n-1]+a_2x[n-2]+\dots+a_Mx[n-M]+b_0g[n]+b_1g[n-1]+b_2g[n-2]+\dots+b_Kg[n-K]$$

自己回帰モデル(autoregressive model)

ARモデルとも呼ばれる．時系列信号において，現在における信号を$x[n]$(n：整数)とし，この信号がある線形システムの出力であると仮定する．このとき，そのシステムの現在の入力信号$g[n]$と，過去の出力信号$x[n-1]$，$x[n-2]$，……，$x[n-M]$の，重み付きの和によって$x[n]$を表すというモデルである．つまり自己回帰モデルでは，信号$x[n]$を次のような式で表現する．

$$x[n]=a_1x[n-1]+a_2x[n-2]+\dots+a_Mx[n-M]+b_0g[n]$$

スペクトル解析(spectrum analysis)

太陽の光はいろいろな周波数をもった電磁波から構成されているが，プリズムを使うとこれを多数の色に分けることができる．光の色は電磁波の周波数に対応しているので，各色の明るさを測定すれば，どの周波数成分がどれだけ含まれているのかがわかる．通常はこれと同様に，注目している信号にどのような周波数成分がどれだけ含まれているかということを調べることを「スペクトル解析」という．

線形予測法(linear prediction)

信号$x[n]$を自己回帰モデルの項に示される式で表現する際に，その係数a_1，a_2，……，a_Mを求める方法が線形予測法である．この係数は，次のようにして求められる．ある時間範囲内でのすべての時刻nに対して，$x[n]$と$a_1x[n-1]+a_2x[n-2]+\dots+a_Mx[n-M]$(実際には入力信号$x[n]$はわからないので，$b_0x[n]$は除く)の差の2乗平均値がもっとも小さくなる

〔図4〕解くべき連立方程式

$$\begin{pmatrix} r[0] & r[1] & r[2] & \dots & r[M-1] \\ r[1] & r[0] & r[1] & \dots & r[M-2] \\ r[2] & r[1] & r[0] & \dots & r[M-3] \\ \dots & \dots & \dots & \ddots & \dots \\ r[M-1] & r[M-2] & r[M-3] & \dots & r[0] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \dots \\ a_M \end{pmatrix} = \begin{pmatrix} r[1] \\ r[2] \\ r[3] \\ \dots \\ r[M] \end{pmatrix}$$

ように，係数 a_1，a_2，……，a_M を選べばよい．なお，このようにして求めた係数は線形予測係数と呼ばれる．

白色化（プリホワイトニング）（prewhitening）

線形予測法を使うスペクトル解析では，スペクトルの概形ができるだけ平坦なほど分析の精度が高くなることが知られている．そこで，線形予測法を使う前に，前もって信号のスペクトルの概形を平坦にする処理を行う場合が多い．この処理を白色化と呼ぶ．スペクトル解析の対象となる信号が音声信号で母音の場合，これを標本化したものを $s[n]$ とすると，よく使われる白色化の処理は次の差分方程式で表されるものである．

$$v[n] = s[n] - s[n-1]$$

フォルマント周波数（formant frequency）

声道（声帯から口または鼻に至る部分）の共振（共鳴）周波数をフォルマント周波数と呼ぶ．人間が発声する際には，声道の形状を変えることによって引き起こされるフォルマント周波数の違いにより，「ア」，「イ」，……といった区別を行っている．なお，フォルマント周波数は，周波数の低いものから順に第1フォルマント周波数，第2……，と呼ばれる．母音を区別するときに重要なのは，第1，第2，第3の三つのフォルマント周波数であるといわれている．

制御シミュレーションの実際

Maple

数式処理言語の一つ．記号が含まれる式の処理が可能であるので，モデリング（微分方程式の導出）や得られたモデルの解析などを行う際に非常に便利なツールである．

MATLAB

行列に対する種々の操作を得意とする数値処理言語．制御系解析・設計用のライブラリ（Toolbox）が充実しており，制御の分野では広く利用されているが，他の多くの工学の分野においても利用可能である．

Scilab

MATLABと同等の処理が可能な数値処理言語．MATLABのように広範な分野のライブラリをもっているわけではないが，制御に関しては十分なライブラリを備えている．最大の特徴は，フリーソフトであること．http://www.rocq.inria.fr/scilab/からダウンロードできる．

Simulink

MATLAB上で動作する非線形シミュレータ．非線形微分方程式に対応したブロック線図を構成することによりプログラミングを行う．そのためのGUIが整備されており，取り扱いが非常に容易である．

Working Model

非線形シミュレータの一つ．実際の寸法，質量などをパラメータとして設定し，物体の動きとしてよりリアルにパソコンの画面上に表示できる．MATLABとリンクし，データの受け渡しを行うことができるため，MATLAB上で設計した制御器を利用してWorking Model上の制御対象を制御でき，制御器の性能を評価できる．

可制御

操作量が状態方程式中のすべての状態に適切に影響を与えることができるかどうかを表す性質．もし，制御対象が可制御でない場合，操作量をいくら増減しても状態量の一部は制御することができない．

極配置法

制御対象の応答特性に強く関係しているのが極である．制御対象が可制御であるならば，状態フィードバック制御を施すことにより，制御対象のもつすべての極を任意の場所に配置できる．そこで，希望する応答から極を特定し，制御した結果の系がそこに極をもつように状態フィードバックゲインを決定する方法が極配置法である．

最適レギュレータ法

「速い応答を実現するためには多くの操作量を必要とする」，といったトレードオフが一般的に存在する．そこで，応答の2乗面積と操作量の2乗面積を足し合わせたものを評価関数にとり，それを最小にするという意味で最適な制御器を決定する方法が最適レギュレータ法である．

状態空間モデル

状態方程式は制御対象の動特性を表現するものだが，制御系を構成する場合，どのような情報をセンサなどを利用して手に入れることができるのかを明らかにしておくことも必要である．これを観測方程式と呼び，状態方程式と観測方程式をまとめて状態空間モデルと呼ぶ．

状態方程式

制御対象に対して導出した連立する高階微分方程式を，適切にベクトルを定義することによって，連立する1階微分方程式に変換できる．このときのベクトルを状態量（状態ベクトル）と呼び，それに関する微分方程式を状態方程式と呼ぶ．

状態フィードバック制御

状態方程式を導く際に登場する状態量は，制御対象のもつさまざまな情報を必要かつ十分に与えるものである．この状態量に基づいて操作量を決定する制御方式が，状態フィードバック制御である．

制御

与えられた対象が「思うがままに動く」ように操作量を加えることが制御の意味するところである．とくに，現在の対象の状態に基づいて操作量を決定する方式をフィードバック制御と呼ぶ．

線形化

どのようななめらかな曲線も，ある点を中心にその近くだけに注目すると，直線とみなせる．一般に，制御対象の動特性を表す微分方程式は非線形特性をもつことが多いが，ある平衡点を中心とした微小な動きに限定した場合，それを線形微分方程式で近似することができる．これが線形化である．

線形制御理論

対象の動特性を理解することが制御の基本だが，その目的で微分方程式が使用されることが多い．一般的には，非線形特性を有するが，それを線形化することにより線形微分方程式に近似できる．この線形微分方程式を対象に制御器の設計を行う理論が，線形制御理論である．

非減衰振動応答

バネにおもりをつけ，少し引っ張った状態で手を離すと，おもりが上下に振動する現象が生じる．これを振動応答と呼ぶ．空気中でそのように振動させると，その振幅がしだいに小さくなって，最終的には停止する．これが減衰振動である．振動が減衰するのは，空気抵抗などがその要因であるため，そのよう

な抵抗がいっさい生じないならば，おもりは永久に振動し続ける．このような減衰のない振動応答を非減衰振動応答と呼ぶ．

平衡点（状態）

対象が静止している状態を表す点．たとえば，時計の振子を考えた場合，振子が鉛直下方（もしくは鉛直上方）にあるとき，外部から操作を加えないかぎりその状態を保ち続ける．

零次ホールド法

線形制御理論に基づいて設計した制御器は，状態空間モデルと同様の構造をもつ微分方程式で与えられる．それをコンピュータ上に実装するためには，差分方程式に近似する必要が生じる．その近似法の一つが零次ホールド法である．

離散系シミュレーションの実際

GPSS（General Purpose System Simulation）

もっとも初期に開発された汎用の離散系シミュレーション言語．メインフレームで稼動し，製鉄工場のプロセスシミュレーションなどに多く用いられた．現在のバージョンはGPSS/H．パソコンで動作する．

HLA（High Level Architecture）

米国DMS（Defense Modeling and Simulation Office）により開発された分散並列処理のためのアーキテクチャ．これにのっとった分散シミュレーションの開発が，軍関係を中心に進められている．

MODSIM III

SIMSCRIPTの開発で著名なCACI社の最新シミュレーション言語．国防総省，IBM社，そしてCACI社で共同開発した離散系シミュレーション言語．オブジェクト指向の要素を採り入れている．

SCS（Society of Computer Simulation：コンピュータ・シミュレーション学会）

コンピュータ・シミュレーションに関する国際学会．機関紙「simulation」を発行．その他，毎年7月にSSC（Summer Simulation Conference），12月にWSC（Winter Simulation Conference）を主催する．

SIMSCRIPT

GPSS，SIMULA67とともに最古のシミュレーション言語の一つ．GPSSがプロセス主体のモデリングであるのに対し，SIMSCRIPTはイベント主体のモデリングに基づく言語の老舗である．システム記述言語の性格に近い．

SIMULA67

GPSS，SIMSCRIPTとともに最古のシミュレーション言語の一つ．また，CLASSの概念など，最初にオブジェクト指向の考え方を取り入れた汎用プログラミング言語．Smalltalkの言語仕様にも少なからぬ影響を与えた．

アクティビティ

離散系シミュレーションを構成する要素の一つ．エンティティがサーバで享受するサービスの行為．サービス開始というイベントでアクティビティが開始し，サービス終了というイベントで終了する．

イベント（事象）

システム内で何らかの状態変化が起こること．エンティティが待ち行列に入る，待ち行列から出る，サーバがサービスを開始する，エンティティがシステムから出る，などあらゆる状態変化はすべてイベントである．

エンティティ

離散系シミュレーションで，系内に存在するあらゆる「もの」を意味する．現金支払機の待ち行列システムでは客や支払機が，LANシステムなら，コンピュータや中を移動する情報パケットがそれぞれエンティティになる．

開放型待ち行列モデル（Open Queueing Network）

待ち行列システムでサービスを受けるものが，システム外から入って，サービスを受けた後にシステム外へ出て行くような形態のモデル．サービスを受けるものの総数は，常に変化する．

シミュレーションクロック

離散系シミュレーションシステム内部での時刻の決定およびその更新機能．離散系シミュレーションでは，イベント（事象）の発生に合わせてそのシステム内の時刻が不連続的に更新される．その時刻を記憶管理する機能．

プロセス

離散系シミュレーションで，あるエンティティに着目したとき，それに関連する一連のイベント列．たとえば客がカウンタAでサービスを受けるとき，客の到着，待ちの開始，サービス開始，サービス終了，客の離脱といった順で発生するイベントを一括して，「カウンタAでサービスを受ける」というプロセスでとらえると，このプロセス連鎖を記述することでシミュレーションモデルを構成できる．

プロセス主体の方式では，モデリングが簡潔でわかりやすいものになるため，多くのシミュレーション言語では，この方式を採用している．

閉型待ち行列モデル（Closed Queueing Network）

待ち行列システムでサービスを受けるもののが，常にシステム内で滞在する形態のモデル．サービスを受けるものの総数は一定数となる．

離散系シミュレーション（Discrete Event Simulation）

システムの状態変化が時間に対して不連続的であるシステムを対象としたシミュレーション．待ち行列システムなどが典型例である．製造システム，通信システムなどを対象にしたシミュレーションは，このタイプのもの．

連続系シミュレーション（Continuous Simulation）

システムの状態変化が時間に対して連続的であるシステムを対象にしたシミュレーション．たとえば，熱源からの熱の拡散移流，気圧勾配による空気流（風）など．多くの場合，微分方程式などで表現したシステムの挙動を数値的に解くことが主体になる．

菊池　誠　大阪大学サイバーメディアセンター
牧野淳一郎　東京大学大学院 理学研究科天文学専攻
吉田たけお　琉球大学 工学部情報工学科
三上直樹　職業能力開発総合大学校 情報工学科
川谷亮治　福井大学 工学部機械工学科
梅田茂樹　武蔵大学 経済学部経営学科

chapter 04

基礎/原理を理解して開発効率の向上をめざす

データベース活用技術の徹底研究

コンピュータ工学の基礎的な柱の一つが，データベース技術である．コンピュータで扱うデータ量はますます増加してきており，より高効率・高機能なデータベースシステムへの要求が高まっている．いままではサーバ系コンピュータ向けの話が多かったデータベースだが，近頃は組み込み機器にデータベース機能を実装するというニーズも出てきている．また，PostgreSQL/MySQLに代表されるような，ライセンスフリーで使えるオープンソース系データベースも普及してきた．

2002年10月号の本誌特集「データベース活用技術の徹底研究」では，まずデータベース技術の基本・原理を解き明かし，組み込み機器にデータベース機能を組み込むのに必要な技術，オープンソース系データベースやOracle/Microsoft SQL Serverなどの商用データベースに実装されている技術，Web情報検索技術を解説した．最後に，演繹データベース，オブジェクト指向データベース，データマイニングといった，次世代のデータベース技術も解説している．

（編集部）

データベース技術の基礎・原理

3層スキーマ(three schema)

データベースの設計や構築において，概念レベル，外部レベル，内部レベルの三つのスキーマを用いる方法．

Access

Windows用の簡易型データベースであり，Officeに含まれる製品である．GUIにより，ユーザーは容易にデータベースを構築することができる．

DB2

IBM社により開発された商用の関係データベースで，現在ではOracleのライバルである．歴史的には，関係データベースの研究はIBM社のCoddのグループで開始された．その研究成果はSystem Rという形で1977年に発表された．これがその後，1980年代になりDB2となった．開発経緯からすると，DB2は関係データベースの本家であり，Oracleが出現するまでのデータベースの中心であったことはいうまでもない．

さらに，DB2ではその時代の最新のデータベース技術が取り入れられている．たとえば，1990年代には，OLAP(On-Line Analytical Processing)と呼ばれる分析ツールや多次元データベースと呼ばれる考え方が出てきたが，これらをいち早く取り入れたのもDB2である．

またDB2は現在，その汎用性を強調するためにDB2 Universal Database(DB2 UDB)といわれている．DB2 UDBの特徴としてデータベース本来の機能であるデータアクセスを多様な形で行うことができる点が挙げられる．DB2 UDBも分散データベースとして構築可能だが，異なるプラットホーム上のさまざまなデータを自由にアクセスすることができる．また，画像や音声などのマルチメディアに対応するために，関係データベースにオブジェクト指向機能が拡張されている．これは，オブジェクト指向関係データベースの実現である．さらに，パフォーマンス向上のためにいくつかの先進的な機能も用意されている．

Excel

Windows用の表計算ソフトウェアだが，データベース機能も備えている．VBAにより，データベースプログラミングが可能となっている．

Oracle

Oracleは，1979年にRelational Software社によりPDP-11上のUNIXで動く関係データベースとして開発された．この会社が，その後のOracle社となった．また，Oracleはメインフレームからパソコンまでの主要なプラットホーム上に実装されている．そしてOracleは商用データベースの中心となった．実際，現在の商用データベースのほとんどはOracleといってもよい．

Oracleの名前を決定的に有名にしたのは，Oracle社が1995年に「ネットワークコンピューティング」というスローガンを出してからである．これはネットワーク自体をコンピュータとみなし，強力なサーバ機能でデータベースを実現するという考え方である．当時はインターネットが流行し始めた頃だが，Oracle社のコンセプトは先見的だった．当然，Oracleはインターネットにも対応した．すなわち，1998年に発表されたOracle 8iでインターネットコンピューティングの機能を追加した（実際，Oracle 8iのiはインターネットを意味している）．なお，現在のバージョンは2001年にリリースされたOracle 9iである．

SQL(Structured Query Language)

関係データベース用の言語．SQLにより，データベースの定義および操作を行うことができる．なお，SQLの命令は，COBOL，C，Javaなどのプログラムに埋め込める．

SQL Server

Microsoft社により開発されたデータベース．パソコンを中心にした事業展開を行っていたMicrosoft社にとって，データベースの開発は急務だった．そして，1980年代にMicrosoft社はSybase社からデータベース技術のライセンスを買い，SQL Serverの開発を行い，1994年に発売した．そして，1996年にはANSI SQLの機能を完全にサポートした．

SQL ServerはWindows上で動く関係デー

タベースであり，分析ツールも装備している．また，GUIベースのユーザーインターフェースとして，SQL Server Enterprise Managerもあり，容易なデータ操作が可能になっている．SQL Serverは，パソコン上のデータベースとしては他のWindows製品と同様に使える．

一貫性(consistency)

データベース中のデータが矛盾せず意味があるということ．一貫性を保持するためには，一つの表には同じタプルがあってはならない．関係データベースでは，特定なタプルを識別する属性を候補キー(candidate key)という．一つの表には複数のキーがある場合もあるが，その中から一つ任意に選んだものは，主キー(primary key)という．

オブジェクト指向データモデル(object-oriented data model)

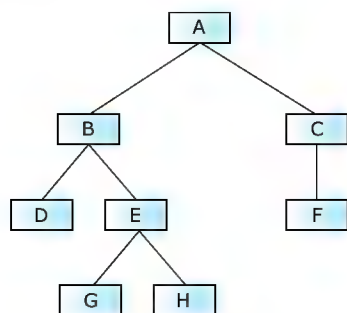
オブジェクト指向に基づいたデータモデルである．データはオブジェクトとして記述され，操作はメソッドで行われる．

階層データモデル(hierarchical data model)

階層的な構造をもつデータを記述するためのデータモデル(**図1**)．階層データモデルは，初期のデータベースでよく用いられた．階層データモデルは，1968年に発表されたIBM社のデータベースシステムIMS(Information Management System)に採用された．

概念レベル(conceptual level)

概念上のデータベースのレベルであり，論理レベルと呼ばれることもある．

〔図1〕階層データモデル

外部レベル(external level)

個人ユーザーの視点のデータベースのレベルであり，ユーザー論理レベルと呼ばれることもある．

関係(relation)

関係とは，直観的にいうと，複数の実体の間に成立する性質のこと．

関係スキーマ(relational schema)

属性名から関係を定義するための概念である．ここで，属性A_iにドメインD_i($1 \leq i \leq n$)を対応させる関数domを，

$$dom(A_i) = D_i$$

と定義する．ここで，属性A_1，A_2，…，A_nをもつ関係Rを$R(A_1, A_2, \ldots, A_n)$が関係スキーマとなる．そうすると，関係$R(A_1, A_2, \ldots, A_n)$は，$dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$の有限部分集合として定義することができる．属性による関係の定義は，上述の定義と本質的には同じであるが，関係データベースの形に近い定義と考えられる．

関係代数(relational algebra)

Codd(コッド)により1970年に提案された関係データベースの代数的な理論的基礎のこと．Coddは，関係データモデルとして関係代数を提案したが，後に関係代数の厳密な形式化を行った．ここで，代数とはある構造に関する操作を記述する理論である．関係代数は関係データベースにおけるデータ操作を形式化できる．また，関係代数は操作的な理論であるため，実際のデータ操作のアルゴリズムと深く密接している．

関係データベース(relational database)

関係データモデルに基づくデータベースであり，リレーショナルデータベースと呼ばれることもある．関係データベースは表として記述されるが，行はレコード(record)，列はフィールド(field)と呼ばれる．現在の商用データベースの中心となっている．

関係データモデル(relational data model)

1970年にCoddにより提案されたデータモデルである．関係データモデルでは，データは関係(relation)と呼ばれる表の形で表現される．そして，関係データモデルは厳密な数学的理論に基づく関係データベースの基礎となっている．関係データモデルでは，データは関係の属性の表として表現される(**表1**)．また，各属性は値をもつが，属性の値の組はタプル(tuple)といわれる．したがって関係データベースは，2次元の表と考えられる．

ここで，「データベース参考書」が関係データベースの名前となる．なお，表の各列はフィールド(field)，各行はレコード(record)と呼ばれる．レコードはタプルとして表されている．

関係論理(relational logic)

関係データベースの論理的基礎であり，関係代数と等価な理論であることが知られている．両者の違いは，関係代数は問い合わせを手続き的に記述するが，関係論理では問い合わせを宣言的に記述する．

実体関連モデル(entity-relationship model)

1976年にChen(チェン)により提案された概念的なデータモデルである．実体関連モデルでは，実世界は実体(entity)とそれらの関連(relationship)により記述される．また，各実体や関連は属性(attribute)をもつ．実体関連モデルは概念的に理解しやすく，関係データモデルに変換可能なため，実際のデータベース設計などにも利用されている．

なお，Chenは実体データモデルを図式化するための実体関連図(ER diagram)も提唱している．実体関連図では，実体は長方形で，関連はひし形で，属性は楕円で表現され各記号は線で連結される(**図2**)．また，関連には1対1関連，1対多関連，多対多関連の3種類がある．

図2は，親子の実体関連モデルを実体関連図で記述したものである．すなわち，実体「親」と「子供」の間には「親子」という関連が存在する．なお，**図2**のように，この親に複数の子供がある場合，「親子」という関連は1対多関連となり，直線上の1とNがその関連

〔表1〕関係データモデル

データベース参考書

著　者	書　名	出版社	出版年
赤間世紀	データベースの原理	技報堂出版	2001
増永良文	リレーショナルデータベースの基礎	オーム社	1990
鈴木健司	データベースがわかる本	オーム社	1998

を示している．

正規化(normalization)

データベース中のデータを理想的な形にするための操作である．正規化では，基本的には次のような規則が用いられる．

(1)関係中では，データは繰り返されない
(2)キーとなる項目とそれらに従属するデータは，別表にする
(3)別表中のデータ間で従属関係があるならば，さらに別表にする

さまざまな形の正規化が提案されているが，一般的には，第1正規化から第3正規化までが必要とされている．

設計(design)

データベースの設計手順は，一般に，次のようにまとめることができる．

要求分析→要求定義→実体関連モデルの記述→実体関連モデルの関係スキーマへの変換→関係スキーマの正規化

知識データモデル(knowledge data model)

人工知能で研究されているデータモデルである．述語論理やフレームなどによるモデルが知られている．

データベース(database)

ある考え方に基づいて，多量のデータをコンピュータに格納して管理するもの．なお，現在ではデータベースはデータベース管理システムと同義に使われることもある．

データベース管理システム(database management system：DBMS)

データベースを管理するソフトウェアのこと．データベース管理システムの必須機能としては，データモデルの実現，データ独立性，データ共有，データアクセス，データ保全が挙げられる．代表的なデータベース管理システムとしては，Oracle，DB2，Accessなどがある．

データモデリング(data modeling)

現実世界のデータのモデルをデータベースのモデルであるデータモデルに変換すること．

データモデル(data model)

データベースのデータを記述するモデルのこと．なお，目的によりいくつかのデータモデルが存在している．おもなデータモデルには，階層データモデル(hierarchical data model)，ネットワークデータモデル(network data model)，実体関連モデル(entity-relationship model)，関係データモデル(relational data model)，オブジェクト指向データモデル(object-oriented data model)，知識データモデル(knowledge data model)などがある．データモデルの種類に対応する異なる種類のデータベースが存在する．

内部レベル(internal level)

物理的なデータベースのレベルであり，物理レベルと呼ばれることもある．人間とデータベースとの関連で見ると，概念レベルは外部レベルと内部レベルの中間のレベルと解釈することもできる．そして，これらのレベルを記述したものをスキーマ(schema)と呼ぶ．

ネットワークデータモデル(network data model)

ネットワーク構造一般を表現するためのデータモデルであり，階層データモデルの改良と考えられる(**図3**)．ネットワークデータモデルは，1964年に発表されたGeneral Electric社のデータベースシステムIDS(Integrated Data Store)に採用された．なお，ネットワークデータモデルと同様のデータモデルは，CODASYL(Conference on Data Systems Language)により仕様化(1971年)されたので，CODASYLモデルともいわれる．

レベル(level)

データベースの設計を行うための階層のことであり，一般には，概念レベル，外部レベル，内部レベルの三つのレベルが用いられる．

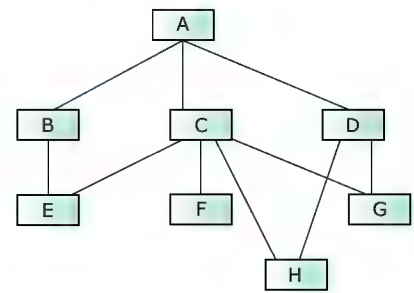
Javaデータベース「PointBase」を使ったモバイルデータベースシステムの構築

ActiveSync

PocketPCなどのWindows CEをベースにしたPDAとWindowsパソコンの間での更新データの同期をおもな目的としたアプリケーション．

Intent

英Tao社が開発した仮想OSである「Elate」をベースにしたJava実行環境．Linuxをベースとしたシャープのザウルス(Zaurus)で採用されていることでも有名である．

〔図2〕実体関連モデル

〔図3〕ネットワークデータモデル

Java

米Sun Microsystems社が開発したオブジェクト指向プログラミング言語．Javaで記述されたアプリケーションは，Java実行環境を配置するさまざまなハードウェア/OSで実行することができる．

JDBC API

Javaプログラムからデータベースにアクセスするための標準API．このAPIによりさまざまなデータベースに同一のマナーでアクセスできる．

JDK1.1(Java Development Kit version1.1)

米国Sun Microsystems社が提供するJavaの開発/実行環境．

Jeode

米Insignia Solutions社の開発したJava実行環境．ガーベジコレクタのしくみが組み込み環境向けに工夫され，PocketPCなどのPDAでよく利用されている．

PDA(Personal Digital Assistant)

おもにハードディスクをもたない小型の情報デバイス全般を指す．アドレス帳やスケジュール管理用のアプリケーションなどが搭載される．

PersonalJava1.2

PDAやセットトップボックスなどのディスプレイをもつ組み込み機器用途を前提として，Javaのバージョン1.1で定義されたAPIのサブセットを定義したものである．

PureJava

OSが提供するネイティブAPIを利用することなく，JavaのAPIのみで記述されたソフトウェアを，PureJavaである，または強調して100％PureJavaであるという．

SQL文

SQLはリレーショナルデータベースで使用される問い合わせ用の言語である．SQL文は英語でいうSQL Statementの訳であり，SQLで記述された問い合せ命令を指す．

SQL92

SQLが広く使われるようになり，ANSIやISOなどにより国際標準規格として仕様が規定された．なかでも1992年に制定されたのがSQL92で，多くのデータベースシステムでサポートされている．

同期機能

二つ以上のシステムの間で，一方で更新された情報を他方に転送して同期させることを指す．双方向に行う場合はとくに双方向同期と呼ぶこともある．データベースシステムのレプリケーション機能も，同期機能の一つである．

オープンソース系データベース「PostgreSQL」の機能と実装

MVCC （Multi Version Concurrency Control，多版型同時実行制御）

トランザクションがある時点でのスナップショットを参照することで，並行性と整合性を保証する同時実行制御方式．ロックを用いる同時実行制御方式と異なり，読み込みと書き込みのロックが競合しないため，問い合わせ（読み込み）と書き込み（更新）が相互にブロックすることがないという特徴をもつ．

PostgreSQL

オープンソースソフトウェアのリレーショナルデータベース．無償で使用でき，BSDライセンスの元での商用利用も可能．副問い合わせ，主キー，参照整合性制約，外部キー，トランザクションといったSQL92の重要な機能がサポートされている．

SQL92

リレーショナルデータベースの標準操作言語で，もっともサポートされているSQL規格．SQL標準というとSQL92を指す．以前の規格SQL86とSQL89はSQL92に包含され，最新規格であるSQL99は，どのリレーショナルデータベースも部分的な実装にとどまっている．

アクセス統計情報

サーバの稼働状況を収集し，報告する機能．タプルアクセスについては，逐次スキャンとインデックススキャンの回数，スキャン・挿入・更新・削除で処理されたタプル数が収集される．入出力については，テーブルとインデックスについて，読み込みブロック数，バッファヒット数が収集される．また，サーバが処理中のクエリーをモニタする機能ももつ．

アクセスメソッド

インデックスでデータをアクセスする方法．PostgreSQLのインデックスは拡張可能なように一般化されている．標準のアクセスメソッドには，B-tree，R-tree，ハッシュ，GiST（Generalized Search Tree）が用意されている．

遺伝的問い合わせ最適化

クエリーの実行は，リレーションを二つずつ結合することを繰り返して行われる．したがって，リレーションの数が多くなるとその組み合わせの数は指数的に増加するため，プランナとオプティマイザの処理に時間がかかる．PostgreSQLでは，リレーション数が11以上の場合は，網羅的な実行計画決定ではなく，結合を対象とした遺伝的アルゴリズムを適用することで，現実的な時間で実行計画を決定する．

エグゼキュータ

プランナとオプティマイザから受け取った実行計画を処理し，問い合わせ結果を返す．実行計画は，クエリーオペレータをノードとする木で，Seq Scan，Index Scan，Sortなどのスキャンとソート，Nested Loop，Merge Join，Hash Joinなどの結合方法のノードがあり，それらのノードが再帰的に処理されて，クエリーが実行される．

オブジェクト識別子OID

データベースオブジェクトを区別するための32ビット符号なし整数による識別子．テーブル，タプル，関数などのデータベース中のオブジェクトに付けられていて，システムカタログの構造の参照値などに使用されている．PostgreSQL 7.2からは，タプルにOIDが付かないテーブルを定義でき，OIDの枯渇を回避している．

オプティマイザ

プランナが作成したクエリーパスをコスト評価し，最適なクエリーパスを予測し，実行計画を決定する．予測には，テーブルデータの行数，データ分布などの統計情報が用いられる．正しい予測のためには，統計情報の適切な更新と分布を考慮した調整が必要．

クエリー

データベース管理システムにおけるデータ定義，検索，更新，削除，データ投入，データ制御などの処理要求の記述．リレーショナルデータベースでは，IBMが開発したSQL言語でクエリーを記述するデータベースが大勢を占めている．

追記型記憶管理

PostgreSQLのデータ記憶方式．更新と追加で，元のデータを直接操作せず，削除マークを付けて残す．簡明な記憶管理方式のため，トランザクション同時実行制御や分離レベル，高速なロールバックなどの実装に役立っている．性能維持のためには，削除領域の回収と索引の再作成を適切に行うことが必要．

データベースクラスタ

データベースを格納する物理領域．一つのPostgreSQLデータベースサーバの管理対象となる．複数のデータベースがデータベースクラスタに保存される．データベースクライアントは，データベースクラスタ内の一つのデータベースに接続して処理を行う．

トランザクション識別子XID

トランザクションを区別するための32ビット符号なし整数による識別子．タプルのシステムフィールドなどで用いられ，同時実行制御などに利用されている．PostgreSQL 7.2からは，2をどのXIDよりも古いものとみなすことと，XIDを剰余で処理することでXIDの枯渇を回避している．

トランザクションログ

データベース更新時に，論理的なデータベース操作を書き込むファイル．同期書き込みなどを使用して，確実に書き込まれる．停電などによる障害でテーブルやインデックスが破壊された場合に，トランザクションログでデータの復旧ができる．

パーザ

クライアントが発行したSQLクエリーを構文解析し，構文が正しければクエリー解析木を作成し，正しくなければエラーを返す処理．

プランナ

パーザとリライタにより作成されたクエリー解析木を受け取り，使用されるリレーションを結合したクエリーパスを作成する．クエリーパスは，リレーションのスキャン方法について，逐次スキャンとインデックススキャンで区別され，結合方法については，入れ子反復結合，マージ結合，ハッシュ結合が区別される．

リライタ

パーザが作成したクエリー解析木を受け取り，問い合わせ書き換え規則を適用して，クエリー解析木を書き換える処理．ビューの実装に使用されていて，ビューへのSELECTは，ビュー定義に置き換えられた後，ターゲットリストやWHERE句による制限などが合成されたクエリー解析木に書き換えられる．

リレーション

データモデルの一つであるリレーショナルデータベースデータモデルでは，データベースを表の集まりとして扱う．この表をリレーションと呼び，そのデータ操作は，リレーショナル代数とリレーショナル論理に基づいて行われる．

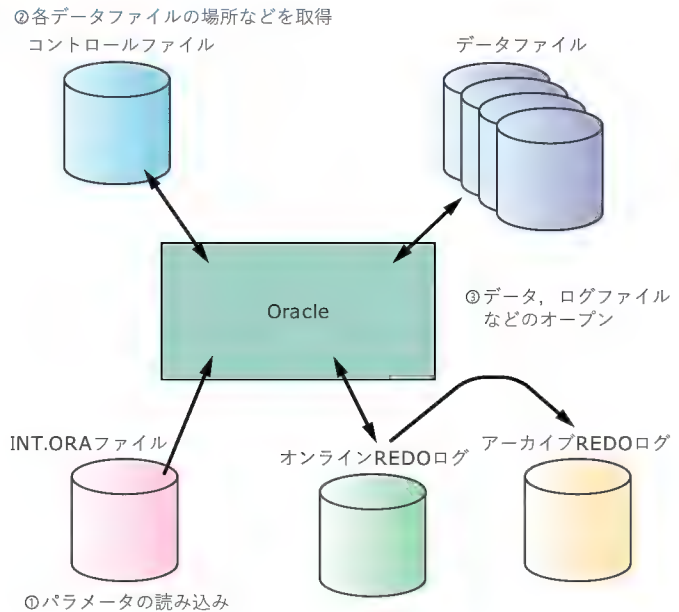
商用データベース「Oracle」で実現されている技術と実装 注1

ACID属性（ACID properties）

1. Atomicity（原子性）成功か失敗のみの結果，2. Consistency（一貫性）矛盾がない，3. Isolation（分離性）別のトランザクションから影響を受けない，4. Durability（持続性）結果が永続的に保証される，の頭字語で，トランザクションの基本属性．

DDL（Data Definition Language）

データベースのスキーマ定義をするための言語．たとえば表を作成するCREATE TABLEなど．DDL以外にDML（Data Manipulation Language），DCL（Data Control Language）がある．

〔図4〕Oracleが使用するファイル

INIT.ORAファイル*

Oracleデータベースのシステムパラメータファイルであり，コントロールファイル名や各種データベースパラメータを定義できる．Oracleが起動されプロセス生成されるとき，INIT.ORAのパラメータを読み込む（**図4**）．

LRU（Least Recently Used）

最近もっとも使われていないものがバッファキャッシュから追い出されるアルゴリズム．頻繁にアクセスされるものがバッファキャシュに残る．効率的なバッファキャッシュの利用を助ける．

Oracle

世界初の商用化されたリレーショナルデータベース管理システム（RDBMS）で，SQLインターフェースをもつ．移植性にすぐれ，多くのプラットホームで実行できる．

SGA（System Global Area）*

Oracleのシステムグローバル領域．Oracleがスタートアップコマンドによりプロセスを開始するときの共有メモリ領域．Oracleの実行コード，共有プール，バッファキャシュ，ログバッファなどがSGAに格納される．

Variable SIZE*

OracleのSGA内のメモリ領域の一つ．共有プール，Javaプール，LARGEプールなどから構成される．たとえば共有プールには，ユーザーが使用したSQLステートメント情報やデータベースのカタログ情報などが格納される．

アーカイブREDOログ（Archive REDO log）**

オンラインREDOログがいっぱいになったときにオンランREDOログの内容をバックアップするファイル（**図4**）．ディスク故障が発生した場合データベースバックアップファイルとともに使用して媒体回復を行う．

アボート（Abort）

トランザクションを終了し，トランザクションで行ったすべてのデータベース操作および保護されているリソースの取り消しを行い，トランザクションの開始点に戻すこと．ロールバックと同義で使われることが多い．

一時セグメント（Temporary segment）*

結合，ソート，副問い合わせなどに使う一時的作業用ディスク領域．読み書きが終了すれば領域は開放される．たとえばソートを行う場合，メモリ内でソートできないときは一時セグメントを使用してディスクソートを行う．

エクステント（Extent）

複数の連続したデータブロックで構成され

注1：この節の見出し横の＊（アスタリスク）について，次のような区別を行っている．無印：一般的なデータベースの説明，＊：Oracleの説明，＊＊：一般的ではあるがOracleを意識した説明．

るデータベース記憶領域の単位．表や索引の作成時，指定されたエクステントサイズが割り当てられ，エクステント内にデータを格納していく．データベースはエクステントがいっぱいになると新しいエクステントを割り当てる．

オンラインREDOログ (On-line REDO log)**

データの更新(追加，変更，削除)時の更新前，更新後，コミット情報など，すべてのデータベース変更情報を書き込むファイル(**図4**)．システムがクラッシュした時，オンラインREDOログを適用しデータベースを復旧する．

強制書き込みバッファ管理 (Force buffer management)

コミットする際，使用されたダーティバッファをすべてデータディスクに書き出す方式．リカバリは単純になるが，コミット時にダーティバッファを書き出すので多くのI/Oが必要になる．

グループコミット(Group commit)

REDOログに複数のコミットトランザクションを1回の書き込みで完了させる方法．同時に複数のトランザクションのコミットレコードもまとめて書き出すことで，REDOログのI/O回数を減らすことができる．

コミット(Commit)

トランザクションがデータベースに対して挿入，変更，削除を行い，その結果を確定すること．コミットを行うことにより，他のトランザクションは更新したデータ内容を見ることができるようになる．

コミット時ログ強制書き出しルール (Force-at-commit rule)

コミット時トランザクション内の変更情報，コミット情報をREDOログに書き出すルール．トランザクションにはREDOログ書き出し後コミット完了を通知する．REDOログに書き出すことで回復が保証される．

コントロールファイル(Control file)*

Oracleデータベースのデータファイル，REDOログのエントリが格納されているファイル(**図4**)．Oracleが起動すると，コントロールファイルを読み各データファイル，REDOログの格納場所を探す．

スチールバッファ管理 (Steal buffer management)

空きバッファが必要になったときに，コミットしていないダーティバッファをディスクに書き出し，該当バッファを別ブロックのために解放する管理方式．

先行書き出しログ・プロトコル (Write-Ahead Log protocol)

バッファの内容を変更し，データファイルに変更の内容を書き出す前に更新前後の情報がログに書かれていることで必ずリカバリできることを保証する規則．WAL(Write-Ahead Log)とも呼ばれる．

ソートエリア(Sort area)**

ソート操作，つまりORDER BY，GROUP BYなどのSQLコマンドを使ったとき内部的に使われるメモリ領域．メモリ領域が足りない場合は，外部記憶装置(磁気ディスク)に書き出される．

ダーティバッファ(Dirty buffer)

ディスクへまだ書き出していない変更済みバッファ．データベースの内容を変更するとき高速化のため最初にバッファキャッシュ内のバッファの内容を変更し，変更されたバッファ内容は非同期にデータファイルに書き出されていく．

チェックポイント(Checkpoint)

バッファキャッシュの内容をデータディスクに書き込み，バッファキャッシュ，REDOログ，データディスクの同期をとること．リカバリはチェックポイントをとったREDOログ地点より回復処理を行う．

データファイル(Data file)**

データベースデータが格納されるファイルであり，一つ以上のファイルが存在する(**図4**)．データの種類として，ユーザーのデータ内容，インデックス，システム情報，ロールバックセグメント，一時セグメントがある．

データブロック(Data block)*

Oracleのデータファイル内のデータが格納される物理領域の最小単位であり，基本的には1回のI/Oの単位，バイトでサイズを指定する．作成時にブロックのサイズを設定する．

動的ビュー(Dynamic view)*

Oracleの内部動作を知る動的性能ビュー．その値はデータベースを使用している間，連続的に更新される．ユーザーはSQLを使用しシステムの性能情報を取得できる．接頭辞V$によって識別される．

トランザクション(Transaction)

コミットまたはロールバックによって終了する回復と同時実行の基本単位．トランザクション内で行ったデータベース操作，およびリソースはトランザクションが終了するまですべて保護される．

パーティション(Partition)

データベースが格納される表や索引を小さな単位に分けること．パーティションに分けることで，データ破壊の局所化，管理単位の細分化，ディスク入出力の最適化，検索の最適化などが可能になる．

ハッシュエリア(Hash area)**

結合(ジョイン)を行いハッシュ結合が選択されたとき，ハッシュテーブルを割り当てるため内部的に使われるメモリ領域．メモリ領域が足りない場合は外部記憶装置(磁気ディスク)に書き出される．

ハッシュ結合(Hash join)

表の結合(ジョイン)方法の一つ．片側の表を読みハッシュ関数により結合キーをハッシュテーブル上に作成し，次にもう一方の表を読み結合キーを同一ハッシュ関数によりマッチさせ(プローブ)結合結果を得る．

バッファキャッシュ(Buffer Cache)

データベースブロックをディスクより読み込み，キャッシュしておくためのメモリ領域．再度同じブロックを読み込む場合，バッファキャッシュからデータを読むことで高速化を行う．

非強制書き込みバッファ管理 (No-force buffer management)

ダーティバッファをデータディスクに書き出す方式の一つ．キャッシュバッファの内容は変更されていようがいまいが，コミットされたかどうかに関係なくキャッシュバッファにとどまり，バッファ交換が必要になったときだけディスクに書き出す．

ページ(Page)

バッファキャッシュにおけるバッファの管理単位．1ページの内容は1ブロックに格納さ

れ，1ブロックは1ページを保持する．ページとブロックは同じ長さである．通常ページとブロックは同意と扱われる．

ページロック(Page lock)

データベースのロック管理をページ単位の粒度で行う方式．並行実行している他のトランザクションが同一ページの別のレコードを更新している場合でもページ全体をロックするので，ロック待ちとなる．

べき等(Idempotent)

操作が一度だけ実行されても，何回実行されても，結果が変わらないこと．検索は何回行っても同じ結果であり，べき等と呼べるが，値に加算していくような更新処理はべき等ではないといえる．

レコードロック(Record lock)

データベースのロック管理をレコード単位の粒度で行う方式．並行実行している他のトランザクションが同一ページの別のレコードを更新している場合，ページ内の該当レコードのみをロックするので待ちになることはない．

ロールバックセグメント(Rollback segment)*

原子性や分離性を実現させるためにOracle内部で使用されるデータ構造．コミットまでの更新前の情報を保ち，アボートが発生したとき，ロールバックセグメントが読み込み込まれ，トランザクションの開始点まで戻される．

ログバッファ(Log buffer)

REDOログに書き出す前に一時的に変更前，変更後情報を格納するメモリ領域．更新系システムにおいてREDOログのI/Oはネックになりやすいが，ログバッファを大きくとることによりI/O回数を減らすことができる．

組み込みデータベースを使ったテキスト全文検索エンジンの実装

AC法(Aho-Corasick法)

テキストのどこにキーワードが出現するかを調べる逐次型の検索アルゴリズムの一つ．テキストを一度読むだけで，複数のキーワードの出現位置をすべて求めることができる．A.V.Aho，M.J.Corasickらによる．

Berkeley DB

Sleepycat Software社が開発・販売しているオープンソースの組み込み向けデータベースエンジン．ライブラリとしてプログラムとリンクされ，APIを通じて，可変長のバイト列の格納や検索の機能を提供する．

B＋木(B＋tree)

ハードディスクのようなブロック単位でアクセスされる媒体に適したデータベースファイルの構成方法．レコード数が非常に大きい場合でも，数回のディスクアクセスでレコードの検索ができる．また，レコードをどのような順序で挿入・削除しても，ブロック内の利用率が50%以上に保たれる．

BM法(Boyer-Moore法)

テキストのどこにキーワードが出現するかを調べる逐次型の検索アルゴリズムの一つ．R.S.Boyer，J.S.Mooreらによる．索引を用いない方法の中では，多くの場合にもっとも高速な方法といわれる．

KMP法(Knuth-Morris-Pratt法)

テキストのどこにキーワードが出現するかを調べる逐次型の検索アルゴリズムの一つ．テキスト中の各文字を一度しか読まないという特徴をもっており，AC法の基礎となっている．D.E.Knuth，J.H.Morris，V.R.Prattらによる．
→AC法

Nグラム(N-gram)

自然言語処理において，言語単位(文字，単語，品詞，音素など)がN個連続したもののこと．$N=1$，2，3の場合はそれぞれunigram，bigram，trigramとも呼ばれる．

PCクラスタ(PC cluster)

比較的安価なPCサーバを構成要素とし，それらをネットワークで結合した並列計算機システムの総称．

SQL(Structured Query Language)

リレーショナルデータベースを操作するための言語．ANSIやISO，JISで規格されており，今日のリレーショナルデータベースシステムの大半に採用されている．

TF・IDF重み付け(TF・IDF weighting)

単語が文書を特徴づける強さを示す指標の一つ．単語の文書内出現頻度(term frequency)に，全文書に占めるその単語が出現する文書の割合の逆数の対数(inverted document frequency)を乗じた値．

Webサーチエンジン(Web search engine)

World Wide Web上に存在するHTML文書や画像を対象とした情報検索システム．

形態素(morpheme)

言語学において，意味をもつ最小単位のこと．

形態素解析(morphological analysis)

日本語などの自然言語で書かれた文を，辞書や文法などを参照して，形態素単位に区切り，それらの品詞を同定する処理のこと．

シグネチャファイル(signature file)

文書中のすべての単語(形態素あるいは文字Nグラム)をハッシュ関数で数値化し，OR演算で結合して得られる固定長のビット列(特徴ベクトル)を並べたもの．単語が出現する可能性がある文書を絞り込むために用いる．

情報検索システム(information retrieval system)

大量のデータの中から，利用者が必要とする情報を含むデータを探し出すシステムの総称．情報検索システムの位置付けを**図5**に示す．文書検索システムや，画像検索システムなどがある．

接尾辞配列(Suffix Array)

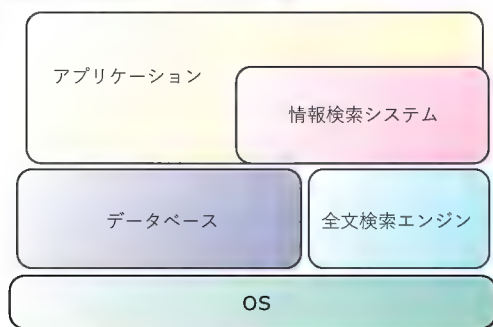
テキスト中のすべての文字を指すポインタを，その参照先の文字列が辞書順になるようにソートした配列(**図6**)．接尾辞配列を二分探索することで，任意の文字列の出現位置を求めることができる．

全文検索(full-text retrieval)

検索用に設定されたキーワードだけでなく，文書中の任意の語句を検索する検索方式．

転置索引(inverted index)

全文検索を高速に行うための索引ファイル構成法の一つ．書籍の巻末索引のように，単語ごとに出現位置のリストを対応づける．日本語では単語ではなく，形態素や文字Nグラムを単位とする．転置ファイル(inverted file)とも呼ばれる．

〔図5〕情報検索システムの位置づけ

〔図6〕接尾辞配列

例："this_is_a_book"の接尾辞配列＝[7, 9, 4, 8, 10, 1, 5, 2, 13, 12, 11, 6, 3, 0]

ポインタ	参照先の文字列
0	this_is_a_book
1	his_is_a_book
2	is_is_a_book
3	s_is_a_book
4	_is_a_book
5	is_a_book
6	s_a_book
7	_a_book
8	a_book
9	_book
10	book
11	ook
12	ok
13	k

→ ポインタを参照先の文字列が辞書順になるようにソートする

ポインタ	参照先の文字列
7	_a_book
9	_book
4	_is_a_book
8	a_book
10	book
1	his_is_a_book
5	is_a_book
2	is_is_a_book
13	k
12	ok
11	ook
6	s_a_book
3	s_is_a_book
0	this_is_a_book

データベース技術の最新動向

ODMG
(Object Database Management Group)

1991年には，オブジェクト指向データベースの標準化を推進するために組織されたグループ．1997年にODMG 2.0と呼ばれる標準化が発表された．なお最新の仕様は，ODMG 3.0となっている．ODMGは，次のような仕様から構成されている．

- オブジェクトモデル(Object Model)
- オブジェクト定義言語(Object Definition Language：ODL)
- オブジェクト交換フォーマット(Object Interchange Format：OIF)
- オブジェクト問い合わせ言語(Object Query Language：OQL)
- 言語束縛(language binding)

OLAP
(On-Line Analytical Applications)

Codd，Codd and Salleyにより1992年に提案されたデータベースの一般化のこと．OLAPで用意されているツールにより，データをさまざまな観点から分析できる．その中の中心的手法は，データを多次元配列で表現するの多次元化である．この考え方に基づくデータベースは，多次元データベース(multi-dimensional database)ともいわれている．OLAPでは，各種の解析ツール群によりスムーズな意思決定が可能となっている．

一階述語論理
(first-order predicate logic)

論理学で研究されている論理システムの一つ．元来，数学で用いられる推論の記述に利用されていたが，人工知能，とくに自動推論の研究の基礎理論にもなっている．実際，定理証明の初期の研究は，一階述語論理の定理証明システム構築が中心課題だった．なお，コンピュータサイエンスの常識であるブール代数(Boolean algebra)は，一階述語論理のサブシステムである命題論理(propositional logic)と等価な代数であることが知られている．

遺伝的アルゴリズム
(genetic algorithm)

生物学で研究されている進化論を応用したアルゴリズムのこと．ある問題の解を遺伝的法則により見つけることができる．遺伝的アルゴリズムでは，最初に異なる遺伝子をもついくつかの初期集団を用意し，選択，交差，突然変異の三つの操作により計算を行う．なお，遺伝的アルゴリズムを応用した問題解決法に関する分野は，進化的計算(evolutionary computation)とも呼ばれている．

演繹データベース(deductive database)

論理プログラムにより記述されるデータベース．論理データベースと呼ばれることもある．演繹データベースでは，データ定義言語とデータ操作言語の両方が同じ論理プログラミング言語となる．さらに，演繹データベースは，関係データベースの拡張と考えることができる．

オブジェクト指向(object-oriented)

1980年代初頭頃から注目されてきたコンピュータサイエンスの一つの方法論．オブジェクト指向では，現実世界を「もの」を表す単位であるオブジェクト(object)が基本となるが，われわれの思考方法にきわめて近い方法論を展開できる．よって，オブジェクト指向はコンピュータサイエンスのあらゆる分野に適用できる．

オブジェクト指向データベース
(object-oriented database)

オブジェクト指向データモデルに基づくデータベース．データはオブジェクトとして解釈されるので，データとその操作であるメソッドをまとめて扱うことができる．したがって，グラフィックスや知識などの複雑な構造をもつデータの管理に適している．

オブジェクト指向プログラミング言語
(object-oriented programming language)

オブジェクト指向をベースにしたプログラミング言語．言語仕様としてオブジェクト指向機能が用意されている．最初のオブジェクト指向プログラミング言語は，Smalltalkである．現在利用されているものとしては，C++やJavaなどがある．

失敗による否定
(negation as failure：NAF)

Clarkにより1978年に提案された，論理プログラミングにおける否定．失敗による否定は，次のように解釈される否定である．すなわち，Aの導出が失敗したならば，not(A)を導出する．失敗による否定は，節のボディ部にnot(B)のように記述することができる．

人工知能
(Artificial Intelligence：AI)

人間の知的活動をコンピュータに実現させるための理論と技術を研究する分野．人工知能は，コンピュータサイエンスの中でももっ

ともむずかしい分野の一つと考えられている．研究の歴史は古く，1950年代中ごろから研究は始まっている．しかし，現在でもその当初の目標が完全に実現されているとはいえず，人工知能実現がいかに困難であるかを示している．

知識工学 (knowledge engineering)

人工知能実現を工学的な立場から研究する分野．実際には，実用的な人工知能システムの開発が目標となっている．

知識ベース (knowledge base)

知識をデータとしたデータベースのこと．エキスパートシステム用のデータベースとして考案された．知識としては事実と規則に対応するものがある．また，知識ベースからの推論を行うための推論エンジンも必要である．

データウェアハウス

データウェアハウスとは，データの「倉庫」を意味する．よって，データウェアハウスは大規模データベースと考えられる．さらに，データウェアハウスでは企業などの意思決定を支援する機能が付加価値として付いている．

データウェアハウスと呼ばれるシステムは，1990年代初頭にPRISM社のInmon（インモン）によって提案された．すなわち，データウェアハウスは過去から現在までのデータを蓄積した大規模データベースを意味し，意思決定において有用な形のデータの処理が可能なシステムである．よって，データウェアハウスはデータベースを発展させた概念と考えられ，意思決定支援システムに付属すべきデータベースの形と解釈することもできる．データウェアハウスでは，データをさまざまな角度から分析し，その結果ユーザーは意思決定を行う．

データマート (data mart)

小規模でローカルなデータウェアハウス．

データマイニング (data mining)

データウェアハウスにおいて基本となるデータ処理技術は，総合的にデータマイニングと呼ばれている．デーマイニングの考え方は，元来統計学に見られていたが，近年では，人工知能（とくに学習），クラスタリング，可視化，データベースなどの分野と関連して論じられている．もっと具体的にデータマイニングを考えると，データの有用な要約の発見法ということになるだろう．また，データベースにおけるデーマイニングに対応する技術は，とくにデータベースにおける知識発見（knowledge discovery in database：KDD）と呼ばれることもある．

データマイニングシステムとデータベースシステムの本質的な違いは，前者ではデータベース中に隠れている知識を取り出し，加工して利用できることにある．よって，データマイニングにより発見された知識を基に意思決定などを行う．データマイニングはデータベースと密接に関連することになる．実際，データベースにおいて，SQLなどの問い合わせ言語を用いると，データベースに関する問い合わせを行える．しかしSQLでは，当たり前の知識を取り出すことはできるが，データベース中に隠れているさまざまな知識を取り出すことはできない．知識発見のためには，さまざまな分野の新しい技術が必要となる．

ニューラルネットワーク (neural netowork)

人間の神経回路をベースにした計算モデルのこと．複雑な人間の学習機能をシミュレートできる．なおニューラルネットワークは，将来実現が期待されるニューロコンピュータの理論的基礎にもなっている．

ファジイ理論 (fuzzy theory)

Zadehにより1965年に提案されたファジィ集合に基づく理論．ファジイ理論では情報のあいまい性を記述することができ，常識推論や制御などに応用されている．

ラフ理論 (rough theory)

Pawlakにより1987年に提案されたラフ集合に基づく理論である．ラフ理論では，情報のあらさを記述することができる．

論理プログラミング (logic programming)

一階述語論理に基づくプログラミングであり，代表的な論理プログラミング言語としてPrologがある．Prologなどの論理プログラミング言語の計算は，J.A.Robinsonが1965年に提案した分解原理（resolution principle）と呼ばれる証明法を基礎としている．

赤間世紀　帝京平成大学 情報システム学科
紅野　進　（株）オージス総研
杉田研治
加藤比呂武　BroadVision,Inc.
原田昌紀　NTT未来ねっと研究所

chapter 05

携帯機器やシステムオンチップで重要な低消費電力/高性能プロセッサ

徹底解説！ ARMプロセッサ

本章では，ARMプロセッサ関連の用語を解説する．これまでの組み込み機器は，ケースを開けてみればどれがCPUなのか，どんなアーキテクチャのCPUが採用されているかという見当がついた．しかしARMプロセッサは，『ARM』と書かれていないものも多く，まして組み込み機器では「○○プロセッサ搭載」というように採用しているCPUを表立ってうたうことは少ないため，ARMは名前の知れわたったプロセッサとはいえなかった．だが現在では，携帯電話やネットワークのルータなど，低消費電力で処理能力も要求される分野でかなりのシェアを占めている．とくにSoC(システムオンチップ)の分野では，大きな存在になっている．

本誌2002年11月号の特集「徹底解説！ ARMプロセッサ」では，ARMプロセッサについて，そのアーキテクチャからプロセッサファミリ，命令セットの解説，プログラミング技法，オプティマイズ事例などを重点的に解説した．

(編集部)

ARMアーキテクチャ詳解

32ビットRISCプロセッサIP

RISC形式の32ビットマイクロプロセッサの提供形態がIPであるもの．ここでのIPとは，設計データを意味する．シリコンチップとしてではなく，設計データとして供給される．製造プロセスや実装の自由度が広いといった利点がある．

AHB
(Advanced High Performance Bus)

SoC内のマクロセル間を接続する標準バスの名称．英国ARM社が仕様を策定し，公開しているバス規格．SoCの内部バスとして事実上の業界標準で，AHBバス仕様のIPが広く入手可能．

ARM11

英国ARM社のARMプロセッサラインナップの最新ファミリ名．アーキテクチャはv6で，マルチメディア処理命令をもつ．製品名としては，ARM1136JF-SとARM1136J-Sがあり，ARM1136JF-SではVFPを搭載する．

ARM6

英国ARM社がスピンオフ後，最初に市場に投入したARMプロセッサ．小さいチップ面積で低消費電力，高性能を実現し，携帯機器に32ビットプロセッサが搭載できることを証明した．米国Apple社のPDA，Newtonに採用されたことで知られる．

ARMの歴史

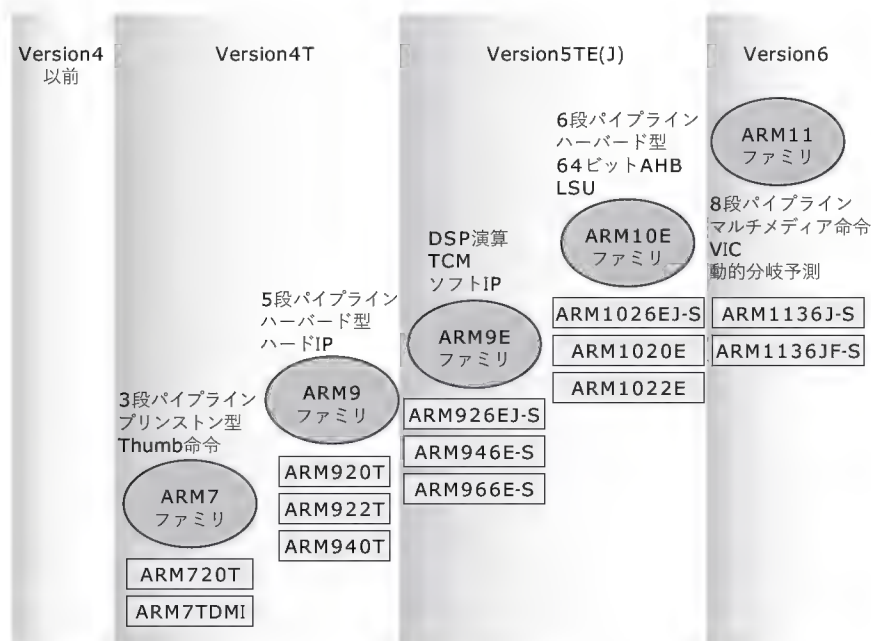
英国ARM社は，1991年にプロセッサ設計会社として，英国Acornコンピュータからスピンオフして誕生した．組み込み市場に特化した32ビットRISCプロセッサの設計会社として一連のARMプロセッサを市場に投入し，大きな市場シェアをもつ．

ARMプロセッサ

英国ARM社の設計する組み込み用32ビットRISCマイクロプロセッサの総称．ARMプロセッサはチップではなく設計データとしてARM社からシリコンパートナーに供給され，そのデータを基にチップが作成される．ARMプロセッサのロードマップを**図1**に示す．

BTAC
(Branch Target Address Cache)

動的分岐予測を行う場合，過去の分岐アドレスと分岐結果を用いて分岐の予測を行う．

〔図1〕ロードマップ

それらのデータを格納するための一種のキャッシュメモリを指す．

CISC（Complex Instruction Set Computer）

日本語では「複雑命令セットコンピュータ」と訳される．RISCプロセッサと比べて命令セットがより複雑で，高度な機能を1命令で実現できるが，プロセッサの動作周波数の向上が難しい．

Embedded ICE

英国ARM社のARMプロセッサに搭載可能なデバッグ回路のIP．SoCに搭載されるCPUの場合，CPU差し替え型のICEが使えないため，最初からICEと同等の機能をもつ回路をCPU内に埋め込んでおく．

ETM（Embedded Trace Module）

CPUに差し替えて使うICEでは当たり前のヒストリダンプ機能は，JTAG経由のデバッガでは実現が難しい．そこで，ARMプロセッサではチップ内にETM回路を搭載し，外部のバッファメモリにサンプルしたヒストリデータを転送する．

Jazelleテクノロジ

ARMプロセッサがJavaバイトコードを直接実行できるようにする技術の総称．ソフトウェアで実現するJavaバーチャルマシンと比較して大幅に実行速度が向上する．携帯電話などで配信されるJavaアプリケーションの実行で効果的．

Memory Protection Unit（メモリ保護ユニット）

メインメモリを複数のブロックに分けてブロックごとに属性をもたせ，メモリアクセスに対して属性の値を評価することで，そのメモリに対するアクセス機能を制限するユニット．

MMU（Memory Management Unit）

MMUの大きな役割は，メモリアクセスの管理と，仮想アドレスから物理アドレスへのアドレス変換の二つがある．アドレス変換は，マルチプロセス対応のOSを実行する際に必須とされる機能．

RISC（Reduced Instruction Set Computer）

日本語では「縮小命令セットコンピュータ」と訳される．CISCプロセッサと比べて命令セットが簡潔で動作周波数の向上がはかりやすい．

SAD演算（Sum of Absolute difference演算）

差の絶対値の合計を求める演算．MPEGのエンコード処理で動き検出をする際，フレームデータ間で行われる演算．

SoC（System On Chip）組み込み用途

ICの大規模化により一つのチップ内にシステムを構成するほとんどのコンポーネントが搭載されるようになり，そのようなチップに向けたコンポーネントを“SoC組み込み用途”という．

TCM（Tightly Coupled Memory）

高速で動作するようにCPUコアと専用バスで直結される小容量のメモリ．キャッシュと異なり，固定的な物理アドレス空間に割り当てられ，ラインフィルは発生しない．

Thumbアーキテクチャ拡張

ARMプロセッサの標準命令セットであるARM命令セットのサブセット版．ARM命令セットでは全命令が32ビット形式だが，Thumb命令セットではすべてが16ビット形式である．Thumbは英語で親指の意味．

TLB（Translation Lookaside Buffer）

MMUが仮想アドレスから物理アドレスに変換する際，メインメモリ中にある変換テーブルの参照動作を高速にするために搭載される，変換テーブル用のキャッシュ．

VFPコプロセッサ（Vector Floating coProcessor）

浮動小数点型のベクタデータの各種演算を実行するコプロセッサ．

インラインシフト

ARM命令セットに特徴的な機能で，データの演算と，演算に用いるオペランドデータのシフト操作の両方が一つの命令で実行できる．通常のプロセッサではシフト命令と演算命令の2命令で実行される．

条件実行

ARM命令セットに特徴的な機能で，ほとんどのARM命令は，条件分岐命令でバイパスしなくても，命令自身が条件に応じて命令の実行をスキップできる．分岐命令によるパイプラインの乱れを回避できる．

静的分岐予測

プログラムの分岐が原因で起きるパイプラインのフラッシュを抑えるため，分岐の方向を予測して，それに基づいてプログラムをフェッチすること．静的分岐予測では予測の根拠は常に決まった規則に基づく．

ダーティデータ

データキャッシュ中のデータがCPUの書き込み動作によって書き換えられることによって，対応するメインメモリ中のデータと内容が同じでなくなったデータ．

遅延分岐

分岐によって発生するパイプラインフラッシュのペナルティを緩和するため，分岐命令で分岐するのではなく，分岐命令から数サイクル後（パイプラインに依存）に分岐すること．その数サイクル間には依存関係のない命令が実行可能．

動的分岐予測

プログラムの分岐が原因で起きるパイプラインのフラッシュを抑えるため，分岐の方向を予測して，それに基づいてプログラムをフェッチすること．動的分岐予測では過去の分岐履歴をBTACに保存し，予測に用いる．

ハーバード型

プロセッサのバスアーキテクチャの一構成．命令メモリとデータメモリを別々にもち，それぞれに対してバスをもつ．命令アクセスとデータアクセスの競合が生じない．

プリンストン型

プロセッサのバスアーキテクチャの一構成．命令とデータを同一のメモリにもち，一つのバスでアクセスする．命令アクセスとデータアクセスの競合が生じるが，メモリは一つですむ．

ライトバッファ

プロセッサ側からメインメモリにデータを書き込む際に，両者の間の動作速度差から生じるプロセッサへの“待ち”を緩和するためのバッファ．FIFOのように動作し，プロセッサ側は低速なメインメモリの動作に制限されなくなる．

ラインフィル

CPUのキャッシュアクセス時にキャッシュ

ミスが発生した場合，キャッシュがメインメモリからミスしたデータを読み出すトランザクションのこと．キャッシュの構成単位であるラインを満たすようにバースト転送が起きる．

ARM命令セットの詳細/ARMプロセッサを採用したシステムの最適化

ATPCS（ARM Thumb Procedure Call Standard）

ARMプロセッサでARM命令およびThumb命令を使用したアセンブラやCなどのルーチンを柔軟に混在し，相互に呼び出しができるように関数の呼び出しと復帰の手続きや，スタックなどのメモリモデルを定めた規格．各レジスタの役割は，**表1**のように定義されている．

アドレッシングモード

命令セットにおいてロード/ストアなどメモリを対象とする操作を実行する場合，プロセッサがその対象アドレスを生成するためのさまざまなルールをアドレッシングモードと呼ぶ．

演算と算術・論理シフトの1命令同時実行

ARMプロセッサの特徴の一つ．通常のプロセッサでは基本演算命令とシフト/ローテート命令は個別の命令として存在することが多いが，ARMプロセッサは一つの命令で基本演算とシフトを同時に実行できる．

演算命令のフラグセット・非セットの選択

一般的なプロセッサでは，演算命令の結果が暗黙のうちにフラグに反映されることが多い．しかし，ARMやPowerPCなどの一部のプロセッサでは，演算結果をフラグへ反映するかしないかを選択できる．これにより，演算を含む制御効率の向上が期待できる．

エンディアンの選択

最近のプロセッサには，プロセッサがビッグ/リトルのどちらのエンディアンで動作するのかを選択できるものがある．このときシステムがどちらのデータタイプを主に扱うかで，プロセッサの使用するエンディアンを決定する．たとえば，TCP/IPのデータを主に扱うシステムではビッグエンディアン，USBのデータを主に扱うシステムではリトルエンディアンをシステムの標準として選択することが考えられる．

外部割り込み

周辺I/Oデバイスなどがプロセッサに対して処理を要求するタイミングが不明確な場合，デバイスから処理が必要な時点でプロセッサに処理を要求する．プロセッサは要求を受け付けることができる状態であれば，現在の処理をいったん中断し，デバイスに対して適切な処理を行った後に中断した処理に復帰する．このようなプロセッサに対する外部デバイスからの要求を知らせるメカニズムを「外部割り込み」と呼ぶ．

コプロセッサ命令

ARMプロセッサなどでは，命令セットや機能の拡張のためにコプロセッサを接続することができる．このコプロセッサに対する命令を「コプロセッサ命令」と呼ぶ．

シャドウメモリ

図2のように，ブート後にI/Oデバイスへの書き込みなどによりROMが配置されているメモリ空間をRAMで置き換える手法を「シャドウメモリ」と呼ぶ．シャドウメモリは，ベクタアドレスが固定されたプロセッサにおいてはシステムの汎用性を高めるために有効な手法である．

乗算命令

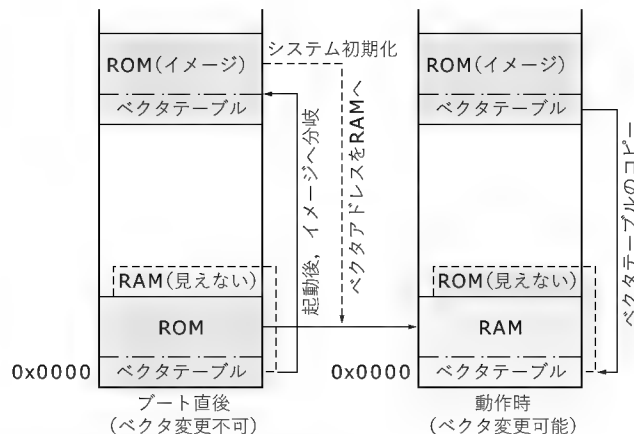
乗算を行う命令．データ処理命令の一種だが，古いプロセッサでは乗算命令をもたない場合が多かったことや，プロセッサ内部で乗算に関して加減算とは異なる回路を使用することが多いため，通常のデータ処理命令とは別に扱われることがある．**表2**にARMプロセッサで使用できる乗算命令を示す．

ステータスレジスタ転送命令

プロセッサでは，プロセッサの実行状態やフラグなどを格納するための特別なレジスタが用意される場合がある．ARMプロセッサではこれを「ステータスレジスタ」と呼び，ステータスレジスタに対する命令を「ステータ

〔表1〕ATPCSでのレジスタ使用目的

レジスタ	別称	別称	役割
R15		PC	プログラムカウンタ
R14		LR	リンクレジスタ
R13		SP	スタックポインタ
R12		IP	ARMステートサブルーチンコール間スクラッチレジスタ
R11	v8	FP	ARMステート変数レジスタ8/ARMステートフレームポインタ
R10	v7	SL	ARMステート変数レジスタ7/スタックリミット
R9	v6	SB	ARMステート変数レジスタ6/スタックベース
R8	v5		ARMステート変数レジスタ5
R7	v4	WR	変数レジスタ4/Thumbステートワークレジスタ
R6	v3		変数レジスタ3
R5	v2		変数レジスタ2
R4	v1		変数レジスタ1
R3	a4		引き数/結果/スクラッチレジスタ4
R2	a3		引き数/結果/スクラッチレジスタ3
R1	a2		引き数/結果/スクラッチレジスタ2
R0	a1		引き数/結果/スクラッチレジスタ1

〔図2〕シャドウメモリ

〔表2〕乗算命令

ニモニック	構 文	意 味	動 作
MUL	MUL{cond}{S} Rd, Rm, Rs	32ビット積算	Rd=(Rm * Rs)[31 : 0]
MLA	MLA{cond}{S} Rd, Rm, Rs, Rn	32ビット積和算	Rd=(Rm * Rs+Rn)[31 : 0]
UMULL	UMULL{cond}{S}RdLo, RdHi, Rm, Rs	符号なし64ビット積算	Rd/Hi : Rd/Lo=Rm * Rs
UMLAL	UMLAL{cond}{S}RdLo, RdHi, Rm, Rs	符号なし64ビット積和算	Rd/Hi : Rd/Lo=Rd/Hi : Rd/Lo+Rm * Rs
SMULL	SMULL{cond}{S}RdLo, RdHi, Rm, Rs	符号あり64ビット積算	Rd/Hi : Rd/Lo=Rm * Rs
SMLAL	SMLAL{cond}{S}RdLo, RdHi, Rm, Rs	符号あり64ビット積和算	Rd/Hi : Rd/Lo=Rd/Hi : Rd/Lo+Rm * Rs

Rd：ディスティネーションレジスタ
Rd/Hi：ディスティネーションレジスタ(上位32ビット)　Rd/Lo：ディスティネーションレジスタ(下位32ビット)
Rm：被乗数レジスタ(32ビット)　Rs：乗数レジスタ(32ビット)　Rn：積に加算する値

スレジスタ転送命令」と呼ぶ.

スワップ命令

ARMプロセッサにおける非同期プロセスやマルチプロセッサ間でのセマフォの実現などに使用する命令. この命令ではリード・モディファイ・ライトの一連の動作が他の要因により妨げられない“アトミック”な動作をシステムが保証する.

データ処理命令

データを処理する命令, すなわちデータに対する加算, 減算やAND/OR/XOR/NOTなどの論理演算, および比較を行う命令. 単に演算命令と呼ぶこともある.

データ転送命令

一般的なRISCプロセッサでは, 演算命令のオペランドはレジスタのみを対象とし, 別途メモリとレジスタ間でデータを転送する命令が用意される. これらのデータ転送, すなわちロード/ストア処理を行う命令をデータ転送命令と呼ぶ. 通常, データ転送命令は対象となるデータサイズ(8ビット, 16ビット, 32ビット, ……)とデータタイプで複数のバリエーションをもつ.

複数レジスタ転送命令

複数のレジスタを一括してメモリに対してロード, またはストアする命令. 割り込み処理の開始・終了時にレジスタ内容の退避・復帰を行う場合や, ブロックコピーなどに使用される. データ転送命令の一種だが, 単純なロード/ストアで構成されるRISC命令セットの中では, 比較的複雑な動作をするため, この命令はデータ転送命令と別に扱われることがある.

分岐命令

プロセッサが処理する命令のアドレスを変更する命令. プログラムで制御構造を実現するためには, 条件付きの分岐命令が必須となる. また, 他のプログラムを呼び出し, 必要な処理の実行後に呼び出した命令の次の命令から実行を再開するサブルーチン呼び出しや復帰命令も分岐命令である.

命令の条件実行

一般的なプロセッサは, 条件付き分岐命令で条件判断を行う. しかし一部のプロセッサでは, ほぼすべての命令実行時に同時にフラグによる命令の実行/非実行(すなわちNOP動作)の決定ができる. これを「命令の条件実行」と呼び, RISCプロセッサでは分岐によるパイプラインの乱れを軽減し, 性能が向上する. ARM, Itanium, 一部のDSP(SHARCシリーズ)などが, 命令の条件実行を採用している.

メモリ構成

システムを構成するメモリの配置されるアドレス位置, バス幅, アクセススピードなどを含む全体的なメモリに関するシステム構成のこと. プロセッサがプログラムコードやワークメモリに正しくアクセスできることをはじめ, キャッシュやライトバッファなどの動作に問題がないこと, またメモリのバス幅やアクセススピードとコストやパフォーマンスとのトレードオフなど, システム設計のうえでメモリ構成を決めることは, もっとも重要な検討事項の一つである.

例外生成命令

プログラム内でソフトウェア割り込みを発生させる命令. ユーザー権限で動作するプログラムが例外生成命令を使用して, より上位権限をもつプログラムを呼び出す動作(システムコール)の実装や, その他ソフトウェアブレークポイントの実装などに使用される.

ARMプロセッサプログラミング事例解説

ARM Developer Suite(ADS)

ARMファミリRISCプロセッサ用アプリケーションを記述・デバッグするためのアプリケーションスイート. 必要なドキュメントとサンプルコードも含まれる.

ARM eXtend Debugger(AXD)

デバッグエージェントを利用してデバッグターゲット上で動作するソフトウェアの実行を検査/制御するためのデバッガソフトウェア. ADSに含まれる.

FIQ

高速割り込み要求.

Integrator

ARMの評価ボードの総称.

IRQ

割り込み要求.

Multi-ICE

マルチプロセッサ・インサーキットエミュレータ. Embedded ICEロジックを含むARMコアを使用したターゲットシステムをデバッグする際に, ホストデバッガとターゲット上のJTAGポート間で使用するインターフェースハードウェア. 接続例を**図3**に示す.

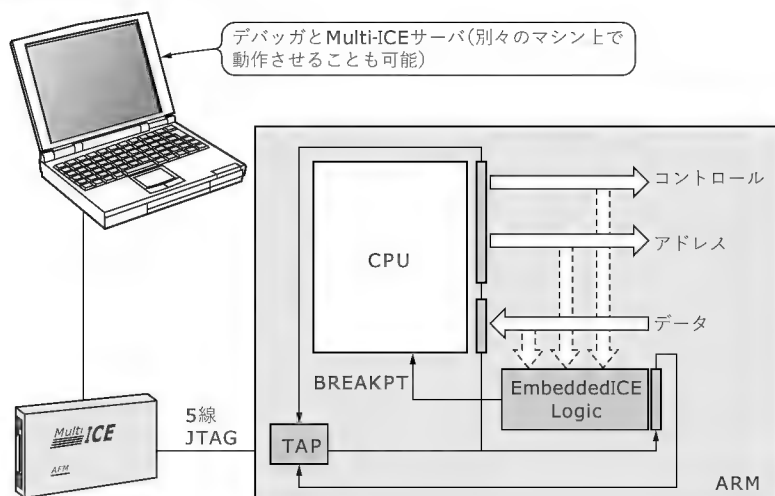
Multi-Trace

ETMを含むARMコアを使用したターゲットシステムのトレース情報を取得するためのトレースアナライザ.

RealView Debugger

マルチコアの混合アーキテクチャのデバッ

Information —— 富士通研究所, ブロードバンド＆ユビキタスラボを設立
(株)富士通研究所は, 富士通の営業や事業部門が顧客から得た要望を元に, 富士通研究所がもつブロードバンド技術とユビキタス技術を利用したプロトタイプを作成する「ブロードバンド＆ユビキタスラボ」を設立した.

〔図3〕Multi-ICEを使ったデバッグ

〔写真1〕RealView ICE

グやOS認識機能などを搭載した，ARMコアベースSoC向けのデバッガ．

RealView ICE

高速コードダウンロード機能，高速ステップ実行機能などを搭載し，デバッグプロセスの改善を可能にしたJTAGエミュレータ(**写真1**)．

Trace Debug Tool(TDT)

ADSの拡張機能で，トレース情報の解凍とプロセッサ動作の履歴表示が可能になる．

スキャッタローディング

アドレスを割り当て，コードやデータセクションを一つの大きなブロックにまとめるのではなく個別にグループ化を行う．

ソフトウェア割り込み(SWI)

この命令(SWI)が実行されるとCPUコアはスーパバイザモードに入り，ソフトウェア割り込み例外を処理する．

ブートコード

リセット時にシステム初期化を行うコード．

フォーマットコンバータ

各フォーマットのデータを必要なフォーマットに変換する機能．

リマップ

メモリマップの変更を可能にする機能．

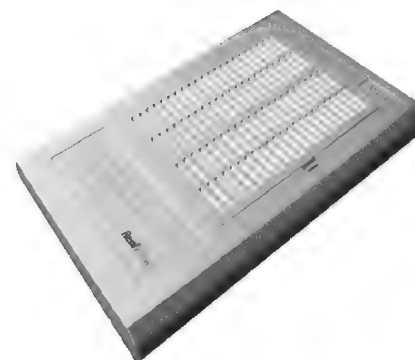
リンカ

アセンブラ/コンパイラが出力した一つ以上のソースオブジェクトから，一つのイメージを生成するソフトウェア．

例外処理

通常のプログラムフローに割り込む内部または外部のイベント．例外が発生すると，プロセッサは現在のプログラム実行を中断して例外ハンドラと呼ばれるルーチンに処理を移す．ハンドラは処理が終了すると，割り込まれたプログラムに制御を戻す．

参考文献

1) *ARM Architecture Reference Manual*
2) *ARM926EJ-S Technical Reference Manual*
3) Steve Furber著/アーム(株)監訳，『改訂 ARMプロセッサ』，CQ出版(株)
4) *Application Note 34 : Writing Efficient C for ARM*
5) *Technical Specifications : The ARM-THUMB Procedure Call Standard*
6) *Technical Reference Manuals : An Introduction to Thumb*
7) *WHITE PAPER : ACCERATING TO MEET THE CHALLENGE OF EMBEDDED JAVA*
8) *WHITE PAPER : The ARM Architecture Version 6 (ARMv6)*
9) *Flyer : The Jazzele Technology/Virtual Machine Interface*
10) *Flyer : PRIMEXSYS WIRELESS PLATFORM*
11) *ADS 1.2 Assembler Guide*
12) *ADS 1.2 Compilers and Libraries Guide*
13) *ADS 1.2 Linker and Utilities Guide*
14) *ADS 1.2 AXD and armsd Debuggers Guide*
15) *ADS 1.2 Debug Target Guide*
16) *ADS 1.2 Developer Guide*
17) *Multi-ICE Version 2.2 User Guide*
18) *MultiTrace Version 1.0 User Guide*
19) *Trace Debug Tools Version 1.2 User Guide*
20) *Integrator/AP ASIC Dev Platform User Guide*
21) *Integrator/CP User Guide*
22) *Integrator/LM-XVC600E+ and LM-EP20K600E+*
23) *APP Note 93: Benchmarking with ARMulator*
24) *ARM9E-S Technical Reference Manual*
25) *ARM946E-S Technical Reference Manual*
26) *ARM926EJ-S Technical Reference Manual*

五月女哲夫 アーム(株) デザインエンジニアリンググループ マネージャ
小林達也 アーム(株) デザインエンジニアリンググループ
織田篤史 アーム(株) フィールドアプリケーションエンジニア

chapter 05

もう日本語対応だけではすまない！

多国語文字コード処理＆国際化の基礎と実際

本章では，ソフトウェア開発を行う際に重要性が高まっている文字コード処理および国際化に関する用語を解説する．2002年12月号特集「多国語文字コード処理＆国際化の基礎と実際」では，文字コードの基礎知識と文字列を扱うライブラリの実例などをくわしく解説した．

最近のソフトウェア開発では，日本語をはじめとした各国語文字列の処理が必須となっている．日本語と一口にいっても，シフトJISだけでなくEUC/JIS/Unicode/ISO10646/TRONコードなど，さまざまな文字コードがある．これらがどういうものかの把握もしないまま開発を進めると，思わぬところで開発がつまずくことさえ起こり得る．また，日本語化にとどまらない国際化のためには，各国語が扱えるというだけではなく，内部的には文字コードに依存しない構造にしておく必要があり，そのための手段はいろいろある．

（編集部）

文字コードの理論と実際/文字コードの変換の実際

ASCII

1963年に米国規格協会によって定められた文字集合/符号化方式．7ビットで表現できる128個（0～127）の文字/制御コードが定義されている．現在の主要な文字コードの多くが，ASCIIをベースに作られている．ASCII文字集合を**図1**に示す．

CJK統合漢字集合

日本，韓国，中国，台湾，香港，米国の6か国とUnicode Consortiumによる漢字統合作業の結果，できあがった文字集合．規格分離漢字規則（各国で別の文字して扱われている文字は別のコードにする）が用いられている．

CP932

マイクロソフトが日本語版Windows用に作成した，シフトJISにNEC特殊文字・NEC選定IBM拡張文字・IBM拡張文字という3種類の文字を追加した文字コード．CPはCodePageの略．

EUC-JP

UNIX系OSでよく使用される文字コード．半角英数字と漢字で利用されるコード領域が完全に分かれているため，プログラムなどで処理が行いやすい．ちなみにEUCには，ほかに韓国語のEUC-KRなどがあるため，EUC-JPを「日本語EUC」ということもある．

ISO10646

Unicodeとほぼ同じもの．もともとはUnicodeとは別物として作業が進められてきた規格だったが，途中でUnicodeを取り入れることになり，現在の形になった．

ISO646

ASCIIと互換性のある独自の文字集合を定義するためのルールを定めた規格．ASCIIの中で，どの文字が変更してはいけなくて，どの文字が変更してもよいのかなどが定められている．

ISO8859

ヨーロッパの文字を表すための文字コード．ISO-8859-1～ISO-8859-16までの15種類（ISO-8859-12は存在しない）があり，それぞれ対応する国/地域が異なる（**表1**）．

JIS X 0201

いわゆる半角英数字，および半角カナを定義した日本語用の文字集合．ISO646の規格に基づいている．ASCIIの文字のうち，バックスラッシュ（\）が円マーク（¥）に，チルダ（~）がオーバーライン（‾）になっている．

JIS X 0208

1978年に定められた漢字などのいわゆる全角文字を集めた文字集合．区点コードとも呼ばれる．漢字（第1水準と第2水準），ひらがな，カタカナのほかに，各種記号やキリル文字，ギリシャ文字なども収録されている．

JIS X 0212

補助漢字ともいう．JIS X 0208に収録されていない文字を補う目的で1990年に作成されたが，あまり普及しなかった．2000年に

〔図1〕ASCII文字集合

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	″	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	＼	l	|
D	CR	GS	−	−	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	／	?	O	_	o	DEL

〔表1〕ISO8859で定義されている文字コード

文字集合名	別　名	対応する言語
ISO 8859-1	Latin-1	西ヨーロッパの言語
ISO 8859-2	Latin-2	中央，東ヨーロッパの言語
ISO 8859-3	Latin-3	南ヨーロッパの言語
ISO 8859-4	Latin-4	北欧の言語
ISO 8859-5		キリル文字
ISO 8859-6		アラビア文字
ISO 8859-7		ギリシャ文字
ISO 8859-8		ヘブライ語
ISO 8859-9	Latin-5	Latin-1にトルコ語を追加
ISO 8859-10	Latin-6	北欧の言語
ISO 8859-11		タイ語
ISO 8859-13	Latin-7	バルト諸国の言語
ISO 8859-14	Latin-8	ケルト語
ISO 8859-15	Latin-9	西ヨーロッパの言語．Latin-1にユーロ記号などを追加
ISO 8859-16	Latin-10	中央，東ヨーロッパの言語

〔表2〕UTF-8の符号化の法則

UCS-4	バイト数	コード範囲
U+00000000～U+0000007F	1	1バイト目：00～7F　0xxxxxxx
U+00000080～U+000007FF	2	1バイト目：C2～DF　110yyyyx 2バイト目：80～BF　10xxxxxx
U+00000800～U+0000FFFF	3	1バイト目：E0～EF　1110yyyy 2バイト目：80～BF　10yyyyxx 3バイト目：80～BF　10xxxxxx
U+00010000～U+001FFFFF	4	1バイト目：F0～F7　11110zzz 2バイト目：80～BF　10zzyyyy 3バイト目：80～BF　10yyyyxx 4バイト目：80～BF　10xxxxxx
U+00200000～U+03FFFFFF	5	1バイト目：F8～FB　111110ww 2バイト目：80～BF　10zzzzzz 3バイト目：80～BF　10zzyyyy 4バイト目：80～BF　10yyyyxx 5バイト目：80～BF　10xxxxxx
U+04000000～U+7FFFFFFF	6	1バイト目：FD～FE　1111110w 2バイト目：80～BF　10wwwwww 3バイト目：80～BF　10zzzzzz 4バイト目：80～BF　10zzyyyy 5バイト目：80～BF　10yyyyxx 6バイト目：80～BF　10xxxxxx

※コードをwwwwwww　zzzzzzzz　yyyyyyyy　xxxxxxxxのように2進数で表現した場合

JIS X 0213が登場したことで，事実上役目を終えた．

JIS X 0213

2000年に策定されたもっとも新しい日本語の文字集合．第3水準漢字，第4水準漢字，①，②などの丸数字，発音記号，音符などが定義されている．

JISコード

正式な名称はISO-2022-JP．エスケープシーケンスを使って文字集合を切り替える「モード切り替え方式」を採用している．すべてのデータを7ビット（0～127）で表現でき，電子メールなどで利用されている．

Unicode

世界中の文字を一つのコードで表そうとするプロジェクト．日本語，韓国語，中国語でそれぞれ利用されている漢字のうち同じものをまとめることで割り当てるコードを減らす「漢字統合」という方法が用いられている．

UTF-16

Unicodeをほぼそのまま利用した符号化方式．2バイト，もしくは4バイト（サロゲートペア）で1文字を表現する．もともとは2バイトですべてを表現できたが，サロゲートペアを採用したために固定長ではなくなった．

UTF-32

Unicodeをそのまま4バイトで表現する固定長の符号化方式．サロゲートペアを採用し，UTF-16が固定長ではなくなったため，Unicode 3.1で新たに策定された．

UTF-8

Unicodeを利用した符号化方式．一つの文字を1バイト～6バイトで表現する可変長のフォーマット．その法則は，**表2**のとおり，ヌル文字も含まず，1バイトで表現する領域はASCIIと互換性があるため，インターネット関連の仕様などで好んで用いられている．

エスケープシーケンス

エスケープ記号で始まる文字列で表現された制御情報．ISO-2022-JPでは文字面の切り替えに利用する．そのためISO-2022-JPはすべての文字を7ビットで表現できるが，エスケープシーケンスの分だけシフトJISなどよりデータが長くなる．

機種依存文字

OSなどが文字コードを独自に拡張して追加した文字．そのOSでしか利用できず，他のOSで表示すると文字化けしてしまうことから，「機種依存文字」と呼ばれる．

サロゲートペア

Unicodeでは16ビットですべての文字を表そうとしたが，それは不可能だったため，一部のコード領域を32ビットのコードとして利用して，利用可能な文字数を増やした．この領域を「サロゲートペア」と呼ぶ．

シフトJIS

マイクロソフトによって策定された文字コード．現在もっとも普及している日本語文字コードである．英数字や半角カナなどの「半角文字」は1バイトで，漢字などの「全角文字」は2バイトで表現される．

住民基本台帳ネットワーク

2002年8月5日に稼動した，住民基本台帳のデータをネットワーク上でアクセス可能にしたもの．戸籍にはさまざまな漢字が利用されているため，「統一文字」という新しい文字コードが策定され，利用されている．
→統一文字

制御コード

文字コードの中には，実際の文字を表すコードのほかに，データの制御のために利用されるコードが定義されている．これらを「制御コード」と呼ぶ．その中にはたとえばタブや改行などが含まれる．

面

文字集合における文字一覧表のこと．縦横に格子状に連なったマス目になっており，そこに文字を埋めていく．一つの文字集合が複

数の面をもつことも可能で，その場合「第0面～第N面」のように表現される．面と文字集合について，**図2**に示す．
→文字集合

〔図2〕面と文字集合

文字コード

コンピュータは数値しか扱うことができない．そこで，文字を扱うためには，利用したい文字すべてに数値を割り当て，その数値の羅列で文字を表現する必要がある．その，文字を数値で表したルールを「文字コード」と呼ぶ．

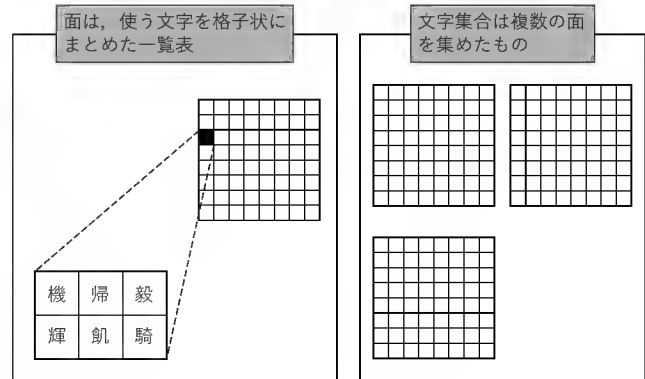
文字コード変換

ある文字コードから別の文字コードに，データを変換すること．同じ文字集合を採用した文字コード同士の変換は簡単な計算でできることが多いが，異なる文字集合を採用した文字コード間では，変換テーブルが必要となる．

文字集合

世界中に存在する文字種の中から，利用する文字を集めた「利用する文字の一覧表」．文字集合と面について，図に示す．
→面

文字化け

文字コードにはさまざまな種類があり，同じ数値に異なる文字が割り当てられているため，ある文字コードで作られたデータを，他の文字コードで表示すると，意味不明の文字の羅列になってしまう．これを「文字化け」と呼ぶ．

符号化方式

文字集合を実際の数値に当てはめるルール．一般に「文字コード」といえば，符号化方式のことをさす．複数の文字集合を同時に使えるようにする場合もある．

変換テーブル

文字コードを変換するために，二つの文字コード間で，それぞれの文字コードが，どの文字コードに変換されるかを記した，一対一の対応表．

多漢字問題とTRONコード

BTRON

TRONプロジェクトのうち，パソコン用のOSやGUI，データ形式，周辺機器（キーボードや電子ペン）などのアーキテクチャ設計を行うのがBTRONである．TRONコードによる多漢字・多言語環境の実現や，障害者向けのコンピュータの操作仕様など，広範囲な内容を含む．

ITRON

TRONプロジェクトのうち，組み込み機器の制御用OSを対象としているのがITRONである．ITRON仕様のOSは，携帯電話，AV機器，家電，自動車などの制御用に圧倒的なシェアをもっている．ITRONの成果を活かしつつ，組み込み機器の開発プラットホームをさらに強化するため，2002年からT-Engineプロジェクトが始まった．

libtf

BTRON上で利用できる，TRONコードを扱うライブラリの名称．tfは"TRON Code Framework"の略．各種の文字コード（シフトJISやEUC-JPなど）とTRONコードとの相互変換機能などをもつ．libtfで提供される関数のリストを**表3**に示す．libtfを含む

〔表3〕libtfで提供される関数

分類	関数	説明
処理環境の管理	tf_open_ctx	処理環境を獲得する
	tf_close_ctx	処理環境を解放する
ID処理	tf_to_id	ID種別とキーワードから，IDを求める
	tf_id_to_idtype	IDからID種別を求める
	tf_id_to_str	IDからキーワード文字列を求める
	tf_id_property	IDからプロパティを取得する
文字コード設定	tf_set_profile	変換元・変換先の外部文字コードを指定する
文字列変換	tf_tcstostr	TC[]を外部文字コードに変換する
	tf_wtcstostr	WTC[]を外部文字コードに変換する
	tf_convtostr	変換元関数から読み込み外部文字コードに変換する
	tf_strtotcs	外部文字コードをTC[]に変換する
	tf_strtowtcs	外部文字コードをWTC[]に変換する
	tf_convfromstr	外部文字コードを変換して変換先関数に書き出す
変換オプション	tf_set_options	変換時のオプションを設定する
文字セット判定	tf_init_charset	文字セット集合情報を初期化する
	tf_query_charset_tcs	TC[]を表現できる文字セットを調べる
	tf_aggregate_charset_tcs	TC[]を表現できる文字セットを，文字セット集合情報に追加する
	tf_aggregate_charset_wtcs	WTC[]を表現できる文字セットを，文字セット集合情報に追加する
	tf_query_charset	TC[]が，文字セット集合情報で表現できるかどうかを調べる

BTRONの開発環境とドキュメントは，「超漢字開発者サイト」からダウンロードできる．

TAD（TRON Application Databus）

超漢字を含めたBTRON上での標準データ形式．TADにより，BTRON上で動作する各種のアプリケーション間のデータ互換性が保証される．

TRONコード

TRONプロジェクトで定めた文字コード体系．150万の文字や漢字を自由に混在して扱うことができ，さらに拡張することも可能．各種の既存の文字セット（JIS，Unicodeなど）を包含している．原規格となる文字セットとTRONコードにおける言語面との対応関係を**図3**に示す．

言語面

TRONコードでは，2バイトで表現される最大48400文字の文字セット（言語面またはスクリプトと呼ぶ）を，必要に応じてエスケープコード（言語指定コードと呼ぶ）で切り替えることにより，個々の文字を特定する．現在のBTRONの実装で利用できる言語面は31面なので，利用可能な文字数は最大約150万字である．

多漢字問題

従来のコンピュータで扱える漢字（たとえばJISやUnicodeの文字）と比較して，より多くの種類の漢字を扱える機能が「多漢字機能」であり，人名用漢字を正確に扱うには重要な機能である．多漢字機能の実現に関する種々の問題が「多漢字問題」である．

超漢字

パーソナルメディアが開発・販売しているBTRON仕様のOS．PC/AT互換機で動作し，OS付属のブラウザ，メール，ワープロ，表計算などのソフトの中で，17万の文字や漢字を自由に混在して利用できる．読めない漢字も簡単に入力できる検字ツールや，文書中の異体字異形字を区別なく検索できる「異形字ゆらぎ検索」（**図4**）など，多漢字を活かす便利な機能が付属．

統一文字

住基ネット上でのデータ交換のために定められた文字の標準規格．JISやUnicodeで表現できない人名用の漢字（とくに異体字）のうち，頻度の高いものを追加収録し，合計2万字強の文字数をもつ．それでも，統 -文字に含まれない人名用漢字は多数存在する．

包摂基準

文字に文字コードを割り当てる際，どのような字体差を区別し，どのような字体差を区別しないかを定めた規則．包摂基準は，JIS，GT書体フォントなどの文字セットごとに異なる．たとえば，JISの包摂基準では1点しんにょうと2点しんにょうの漢字（例：「辻」と「辻」）は区別されず，同じコードが割り当てられているのに対して，GT書体フォントでは両者を区別して別々のコードが割り当てられている．正しくは「包摂規準」と書く．

Qtをサンプルとした国際化の実例

ISO-8859-*

ヨーロッパなどで使われている文字コード．現在ISO-8859-1～ISO-8859-15まであり，ASCIIを基本として0xA0～0xFFにアクセント記号付きの文字や各国・各言語に固有の文字を割り当てたもの．

KOI8-R

ロシア語のキリル文字を表す文字コード．デファクトスタンダードとして広く使われており，RFC 1489にも登録されている．

KDE（K Desktop Environment）

UNIX/X11上で動作するオープンソースのデスクトップ環境．ベースとなるGUIツールキットにQtを使用している．

Qt

Troll Tech社が開発している商用のマルチプラットホームGUIツールキット．Windows，UNIX/X11，Mac OS Xに対応している．UNIX/X11版は，QPLまたはGPLにしたがった非商用のオープンソースソフトウェアの開発に使用する場合にかぎり，無料で使うこともできる．

〔図3〕各種の文字セットとTRONコードとの関係

TRONコード

第1面(FE21)	第2面(FE22)	第3面(FE23)	第4面(FE24)
JIS X 0208, X 0213, X 0212 GB 2312, 点字 KS X 1001	GT書体フォント(1)	GT書体フォント(2)	予約
第5面(FE25)	**第6面(FE26)**	**第7面(FE27)**	**第8面(FE28)**
予約	CNS 11643 -1986(Big5)	予約	大漢和辞典収録文字
第9面(FE29)	**第10面(FE2A)**	**第11面(FE2B)**	**第12面(FE2C)**
大漢和辞典収録文字 記号類	トンパ文字など	予約	予約
第13面(FE2D)	**第14面(FE2E)**	**第15面(FE2F)**	**第16面(FE30)**
予約	予約	予約	Unicode非漢字*(1)
第17面(FE31)	**第18面(FE32)**		**第31面(FE3F)**
Unicode非漢字*(2)	予約	…予約…	予約

一つの箱が最大で48400字

*CJK統合漢字とハングルシラブルは除外

※第N面のあとの(～)内は言語指定コードを表す

Qt/Embedded

Qtの組み込み版．Qtのプラットホーム依存部分として，X11なしのLinuxをサポートしたもの．GUIツールキットの中で使わない機能を外すことにより小型化できる．

Qtopia

Qt/Embedded上で動く，Palm風のPDA向けアプリケーション環境．メニューやPIMアプリケーションなどが含まれている．シャープのLinux版Zaurusでも使われている．

Unicode

世界中の文字を一つのコード体系で扱うことをめざして作られている文字コードの規格．現在の最新版はUnicode 4.0．関連して，プログラムで文字を扱う際に参考にしたり，従うべき技術レポートも多数発行されている．

X11

X Window System，Version 11のこと．

XFontSet

X11が提供している国際化対応のライブラリを使う場合に使用するフォント．複数の文字セットのフォントをまとめて扱うことができる．たとえば，日本語の場合はISO-8859-1とJIS X 0201とJIS X 0208の三つを合わせて使う．

XIM（X Input Method）

X11が提供している国際化対応のライブラリでの文字入力フレームワーク．かな漢字変換など，ユーザーが入力したキーをアプリケーションに渡す前にXIMサーバでハンドリングすることができる．

XPG4
（X/Open Portability Guide，Issue 4）

UNIXおよびC言語における国際化機能に関する仕様．ロケールやマルチバイト文字列・ワイド文字列を扱う関数，コマンド，ユーティリティなどについて規定している．

〔図4〕超漢字4の異形字ゆらぎ検索（実身仮身検索）と検索オプション

X Window System

ネットワーク透過なウィンドウシステム．UNIX系のOSではデファクトスタンダードのウィンドウシステムで，商用UNIXからフリーのUNIXまで広く使われている．

ウィジェット

GUIにおける部品のこと．ラベルウィジェットやボタンウィジェットなど．OSによっては「コントロール」と呼んでいる場合もある．

日本語EUC

日本語の文字列のエンコード方式の一つ．ASCII文字は最上位ビットが0，日本語の文字は最上位ビットが1となっている．マルチバイト文字列として扱うことができる．「EUC-JP」ということもある．

マルチバイト文字列

1文字が1バイトだったり2バイトだったりそれ以上だったりするような文字列．プログラムの中で文字単位の処理を行おうとすると，とても手間がかかる．シフトJISや日本語EUCなど．

文字列編集ウィジェット

キーボードから文字を入力し，文字列への文字の追加や削除などといった編集を行うためのウィジェット．多国語対応のものを作るのはとてもたいへんである．

ワイド文字列

一つの文字を`wchar_t`型で表し，文字列を`wchar_t`の配列として表したもの．プログラムの中で文字単位の処理を行うのがとても簡単である．通常は，プログラムの内部でのみ使用される．

水野貴明
松為 彰 パーソナルメディア（株） 企画本部
高木淳司

chapter 01

オリジナルアーキテクチャのパソコンを作ろう！

作りながら学ぶ コンピュータシステム技術

PC/AT互換機のCPUが386だった頃は，マザーボードにも74シリーズのTTLがよく使われていた．しかし現在では高速化/高機能化が進み，あらゆる機能がSuperI/Oなどのチップセットに集積されるようになっている．また，組み込み向けCPUにおいても，SDRAMコントローラやPCIバスコントローラが内蔵され，リファレンスマニュアルどおりにピンを接続してレジスタを初期化するだけで，動くシステムができてしまう．これでは，本当の意味でSDRAMやPCIを理解することはできない．

そこで本誌2003年1月号の特集「作りながら学ぶコンピュータシステム技術」では，コンピュータシステムを構成する各種要素技術を，実際に試しながら学べるよう，FPGAで各種ハードウェアを実現した．ホストCPUにはSH-4を採用し，ローカルバスをFPGAに直結する．メインメモリとしてはSDRAMコントローラを，I/O拡張インターフェースとしてはPCIホストコントローラを設計する．そしてPCIバス上に，グラフィックス表示，キーボード&マウス入力，ストレージインターフェースを実現した．

本章では，上記特集全体に関する用語をまとめて解説する．（編集部）

英数字・記号

10K/10KH
モトローラ社が製造するECLロジックの10K番/10KH番台の略称．このほかECLinPS（エクリンプスと呼ぶ）などのファミリもある．

4000番台
TTLの74ファミリと同様にメタルゲートCMOSのロジックファミリ群である4000番台の総称．TTLに比べてはるかに高耐圧であるため，現在でも現役．

54ファミリ
74ファミリと同様の論理IC群．ミリタリ/航空/宇宙，また高信頼性システム向けのロジックファミリであり，74xxファミリと多くの互換製品をもつ．

74ファミリ
TTLロジックファミリ群の代表格である7400番台の総称．TI社，NS社やOnSEMI社（Motorola社から分社）などの複数社から供給される，ロジック製品の代表格．

ATA/ATAPI
ATAとはAT Attachiment，ATAPIはATA Packet Interfaceの略．IDEの登場当初は各メーカーのHDDとホストとの間で互換性が確保されていなかった．そこでANSIにおいて互換性を高めるために規格を規定したのがATAである．ATAPIは，ATAインターフェースのうえでCD-ROMドライブを扱うための規格．

BitBLIT
ビットブリットと呼び，グラフィックスにおいてある領域の矩形を転送する場合の呼び名である．矩形転送を行う回路を「ブリッタ」と呼び，多くの場合GPU内蔵DMACで行われるため，CPUによるアクセスに比べ，格段に速い．

Bpp (Bit per pixel)
とくにグラフィックス系では表現できる色の分解能を示す．24Bppというと，R/G/B各8ビットを表す．

CASレイテンシ
SDRAM系で，CASコマンドを発行してから実際にデータが出力されるまでのレイテンシ＝遅延時間．CLと表記される．一般的に高速メモリになるにしたがい（例：SDRAM→DDR-SDRAM），CASレイテンシは多くなる．

CMOS
NチャネルとPチャネルを組み合わせて（＝Complex）構成したMOS-ICの総称．今日ではほとんどのロジックICがCMOSプロセスで構築されている．

CPCI/CompactPCI
信頼性の低いカードエッジ基板のコネクタ部分をDIN形式のコネクタに切り替え，より信頼性を高めたPCIバス規格．VMEバスシステムと同一のラックが使えるようなサイズになっており，VMEバスから移行しやすい形式になっている．制御方式や規格自身は，PCIバスと同一．

CPLD (Complex PLD)
IC内部にPALやGAL構造をもつ機能ブロックを複数個包括したデバイスを示す．FPGAよりは小規模で，SPLDよりは大規模なICファミリ．

DDR-SDRAM
動作クロックの立ち上がり/立ち下がりの両エッジを使ったデータ転送を行う，クロック同期式SDRAM．従来のSDRAMに対して（データのみ）約2倍のデータ転送が可能である．

DIMM (Dual Inline Memory Module)
JEDECで規定された64ビットのデータバスをもつメモリモジュールの総称．SDRAMやDDR-SDRAMを搭載したメモリモジュールが一般的．

DLL (Delay Locked Loop)
遅延素子を駆使して外部信号とIC内部で

信号の変化するタイミングが一致するように制御する機能．PLLと同じ意味で使われる．

DMAC
(Direct Memory Access Controller)

DMA機能を行うコントローラの略称．日立製作所のHD68450(Motorola MC68450)やIntel社の8237が一般的に知られている．

DMA転送

DMACによるデータ転送を示す．アドレス境界や転送長をチェックしながら行うCPUによるデータ転送と違い，DMACにより全自動で行われるので高速．

ECC

エラーチェック/コレクションを行う制御機能．パリティ機能よりもさらに高度な演算を行い，エラー検出と訂正を同時に行える．

ECL

バイポーラトランジスタで構成された，エミッタ出力のロジックファミリ．消費電力は大きいが，非飽和領域でトランジスタを駆動するためにTTLの数倍の速さで駆動することができる．

FASTBUS

IEEE960-1986で制定された，核物理学研究機関システム用の超高速システムバス規格．ECLインターフェースが使われ，マルチバスマスタシステムと高度な割り込み通信系をもつ．ただし，負電圧のECLを使うために，VMEバスと違って一般には広まらなかった．

FCRAM

富士通が開発した，ランダムアクセスのオーバヘッドを少なくした低消費電力のDRAMベースメモリ．外部インターフェースによって疑似SRAMやSDRAMと合わせたものが製品化されている．とくに疑似SRAM形式のFCRAMは，リフレッシュ動作が不要で大容量かつ簡易に使えるため，モバイル/PDA系に適している．

FIFO
(First In First Out)

先入れ/先出し方式のバッファ機構．ちょうどパイプのように，片方から送られたデータはもう片方の出口から出てくる．速度差のあるシステムバス間などの通信に用いることで，性能を改善することができる．

FPGA
(Field Programmable Gate Array)

主としてRAMベースの機能ブロックを集合した，任意の時点で機能書き換えができるPLDの総称．

fps (Frame per Second)

リフレッシュレートを数値で表す場合の単位．1秒間あたり何フレームを表示するかを示す場合に用いられる．LCDなどでは60fpsが一般的．

GAL
(Generic Array Logic)

MMI社のPALに対抗し，Lattice社が開発したEEPROMプロセスベースの，書き換え可能な小規模PLDファミリの総称．

GPU
(Graphics Processing Unit)

CPUやDSPに対して，PlayStation2のGSエンジンなどのグラフィックス機能に特化した画像処理や信号処理を行う高性能プロセッサ．

H8マイコン

日立製作所社製の1チップマイコンファミリ．8ビットマイコンながら低価格と高性能なためにエンジン/モータ制御や入力系制御などに広く使われている．

I/Oバス

外部の制御チップとのInput-Output(入出力)を司るバスを示す．PCIバスシステムでは，バス速度に応じてCPUやメモリなどの「ローカルバス」とSuperI/Oチップの「I/Oバス」に分かれる．

IACKxx

VME，M68Kファミリの割り込み応答サイクル全般の動作を示し，番号が付帯する．たとえばIACK7というと，割り込み優先順位7(最高位)割り込みに対しての応答を行ったことを示す．

IDE〔Integrated Drive(またはDevice) Electronics〕

16ビットデータバスに，チップセレクト2本，アドレスバス3本，I/Oリード(DIOR#)やI/Oライト(DIOW#)などの信号線を使ってHDDを読み書きする．ホストとデバイスはマスタ/ターゲットの関係であり，ホストからのコマンドにデバイスが応答するという方式を採る．そのためインターフェース回路が簡単であり，PC/AT互換機の標準HDDとして採用され普及した．

INT*/IRQ*

割り込みレベルを表すことのできるIPL*に対して，単純に割り込みが発生したことだけを示す単一の割り込み要求ライン．Pentium/PowerPC系は割り込み入力が1本しかない．

IPL (Initial Program Loader)

初期プログラムロードライブラリ．BIOSによりシステム初期化後に駆動され，HDDやCD-ROMなどの読み込み開始を行う．

IRLxx (Interrupt Request Level：割り込み優先順位信号)

SH-4やM68Kアーキテクチャなどにある信号線で，外部割り込みの重要度レベルをエンコードした信号．

IRQxx

割り込みのレベルを説明するときに使用する言葉．割り込みレベルが16あるとすると，通常レベル15が最高位，レベル0が最低位になり，IRQ15などという．

IRQ共有割り込み

同一の割り込み信号ラインや割り込みレベルを共有して使用する割り込み方式．少ない本数で多くの割り込み要求に対応するための苦肉の策であるが，ソフトウェアの応答性能が悪くなる諸刃の剣．

JEDEC
(Joint Electron Device Engineering Council)

アメリカに拠点を置く，電子部品の標準化を規格化する団体．

LIFO (Last In First Out)

後入れ/先出し方式のバッファ機構．スタック(STACK)と同一の機構だが，FIFOに対してLIFOという言葉で呼ばれる．

LVCMOSレベル

3.3V駆動のCMOSデバイスを使用する際のインターフェースレベルを表す．一般的に信号振幅は“H”レベル＝V_{CC}×90%以上，“L”レベル＝V_{CC}×10%以下であり，ノイズマージンが大きい．

● LVTTLレベル
3.3V駆動のTTLデバイスを使用する際のインターフェースレベルを表す．一般的に信号振幅はTTLと同一の"H"レベル＝2.0V以上，"L"レベル＝0.8V以下である．

● M16C/62
三菱電器社製の16ビットマイコンファミリ．そのうち，内蔵ROMが128Kビットあるものが M16C/62である．タイマやUART，PWMコンバータやA-Dコンバータが入っている．

● M32R
三菱電器社製の32ビットマイコンファミリ．組み込みマイコンであるM16Cファミリとは違い，カーナビゲーションシステムなどのCPUに利用される．

● M68K系アーキテクチャ
米モトローラ社から発売されている32ビットプロセッサのアーキテクチャ．初期のワークステーションに採用されていたMC68000，MC68020やMC68030，そして携帯基地局にも搭載されたハードワイヤ化のMC68060がある．現在同社はM68Kアーキテクチャ以外にPowerPCアーキテクチャやARMアーキテクチャなども手がけている．

● MBRAM
マルチバンクRAMの略で，複数の記憶領域を「バンク」という概念で接続して同時に複数動かすことで速度を向上させたメモリ．またメモリを細かいバンクに分けておくことで非動作バンクは休止できるため，低消費電力化も可能となる．

● MC68000
初のM68Kアーキテクチャを実現し，1979年に登場したプロセッサの名称．内部32ビットレジスタ長であり，16Mバイトのリニアなアドレス空間，優先順位付き割り込み応答，豊富な命令やシステム保護機構の搭載などで，他のCPUを圧倒する優れた機能が搭載されていた．

● MC68030
昨今はあたりまえになったが，CPUコアにメモリ管理機構のMMUとキャッシュユニットを統合した第3世代のM68Kアーキテクチャプロセッサ．内部/外部ともに完全な32ビット構成であり，多くのワークステーションや高価なパソコンに搭載された実績をもつ．

● MECL
各社から発売されているECLデバイスで，とくにモトローラ社ECLデバイスファミリの総称を示す．10K，10KH，100KやECLinPS/Liteファミリなどがある．

● MIL-883C/D
アメリカ米軍の半導体製品規格の一つであり，54ファミリのように高信頼性システムの検査規格．

● MIPS系アーキテクチャ
パイプライン構成を改善し，簡素な回路構成と高度なコンパイラ技術によりハード/ソフトの両面からプロセッサの機能向上をめざすRISCプロセッサのアーキテクチャ総称．MIPS3K/4KファミリはWSや組み込み機器だけでなく，家庭用ゲーム機器などでも広く使われている．

● MMU（Memory Management Unit）
メモリ上のOS格納領域やシステム保護領域に対して，アプリケーションソフトなどからのアクセスをハードウェア的に禁止するメモリ管理機構を提供する機能/ICの総称．メモリ書き込み保護やI/Oデバイス群の読み出し保護だけでなく，未実装メモリへのアクセスを検出してOSなどによる仮想メモリ実現のための手助けも行える．

● MPXモード
SH-4の外部バスアクセス方式の呼称で，アドレスとデータを時分割した「マルチプレクスバス方式」を示す．

● NECL
一般的にECLデバイスは負電圧で駆動するが，正電圧で駆動するPECLデバイスの登場により，既存のECLをNegative-ECLとして呼ぶ．

● nXXXX
「＊」信号名（p.84）と同意だが，一般的な回路図エディタでは「＊」は信号名として使えないので，負論理を示す「n：Negative」を信号名先頭に付加することで対応する．

● PAL（Programmable Array Logic）
「パル」と呼ぶ．旧MMI社（現AMD社）のPAL16ファミリが代表的．ワンタイムヒューズ方式で数百ゲート規模の容量をもつPLD．

● PALCE
1回しか書き込みできないPALデバイスに対して，EEPROMプロセスを利用して幾度もの書き換えに対応した小規模PLDデバイス．

● PC100
JEDECのメモリ規格で規定された，バスクロック100MHz駆動のDIMMメモリ規格．ピーク性能時で800Mバイト/秒のデータ転送能力をもつ．

● PC133
PC100と同様だが，こちらはバスクロック133MHz駆動になっており，それによりピーク性能時で1Gバイト/秒のデータ転送能力をもつ．

● PCG（Programmable Character Generator）
グラフィックス性能に乏しかった古いパソコンで，絵を出すために「A」や「B」などの文字コードに絵を当てはめ，それを組み合わせてグラフィックスを表示する方式．その後スプライトに置き換わった．

● PCI（Periheral Component Interface）バス規格
コンピュータを構成する周辺デバイスのデータ転送方式を規定した規格の総称．PCI-LocalBus規格やPCI-Bridge規格，PCI-BIOS規格などの複数の規格から成り立っている．

● PCI/シングル転送
PCIバスの転送方式で，1回のデータ転送のみを行う転送方式．PCIバスはマルチプレクスバスのため，シングル転送は転送効率が悪く（ライト時，最短3クロック），バースト転送が主として用いられる．

● PCI/バースト転送
PCIバス上でクロックに同期して連続にデータ転送を行う転送方式．最初の1ワード目のみオーバヘッドがあるが，以降はオーバヘッドなしにデータ転送が行えるため，最大の転送速度を得ることができる．

● PCIバスブリッジ
システム上の既存PCIバスをプライマリバスとして新たにPCIバスを追加する機能をもつデバイス．本デバイスがブリッジ＝橋渡しを行い，セカンダリバスを追加する．

PCM (Pulse Code Modulation) サウンド

PCM＝ある時間ごとにディジタル符号化したデータを音楽に適用したもの．CDやWindowsのWAVファイル，駅の音声案内などのほとんどがPCMサウンド方式である．

PECL

正電圧側でECLを駆動させる場合の呼び名．当初はPseudo(疑似)-ECLだったが，いつのまにかPositive(正論理)-ECLの略になってしまった．

Pentium

Intel社が製造する，x86アーキテクチャに基づく最新プロセッサファミリ群の総称．8086から続く長い歴史をもち，下位命令は完全に互換である点が他社製品に対して特徴的である．

Pentium4

Intel社が製造するx86アーキテクチャの最新デバイスファミリ．既存のx86に比べて高クロックでの駆動が見込まれる回路構成になっている．

PICMG仕様

PCIバスを工業製品に適用すべき目的で作られたグループが継承する仕様．PCIカードエッジ形状にPCIホスト機能の信号線などを盛り込んでおり，周辺にパソコンなどのPCIバス製品がそのまま流用できるため，製品の入手もしやすい．

PIO転送

DMA転送に対して，1ワードずつのデータ転送をCPUが行う方式を示す．とくにATA/ATAPIやIDEなどのアクセスで使用される言葉である．DMA転送に比べて転送速度は低いが，機種依存性がなく，確実に動くという保証がある．

PLL (Phase Locked Loop)

外部クロックと内部発振回路の位相を検出して，位相をロックすることでICの外部と内部のクロックの差をなくす機能．

Powerアーキテクチャ

IBM社のRISCプロセッサであるRS 6000の意思を受け継ぎ，大型コンピュータからパソコンまで一貫して利用できるように考え出された64ビットのRISCプロセッサアーキテクチャ．本アーキテクチャを継承した製品が，同社とMotorola社から発売されている．

Power4

IBM社がハイエンドサーバ向けに開発した64ビットPowerアーキテクチャプロセッサ．マルチプロセッサ構成を主とした使い方になるよう最適化された，Powerアーキテクチャの頂点に立つハイエンドプロセッサ．

PowerPC

パソコン向けに改良された32ビットPowerアーキテクチャをもつプロセッサの総称．PCはパーソナルコンピューティングとも呼ばれる．第4世代PowerPCプロセッサでは内部レジスタ構成が32ビットになっているだけで，他の部分はPowerアーキテクチャとそう変わらない．

PowerPC/G4

PowerPCアーキテクチャの第4世代(4th-Generation)の略．Motorola社より発売中のPPC75xxでは，既存G3に対して，ベクトル演算ユニットの追加やL2/L3キャッシュの搭載，マルチプロセッサ構成が可能などと，大幅に性能が向上している．

L1/L2/L3キャッシュ

複数の速度で動くキャッシュシステムを階層的に説明する言葉．数字が小さいほうが高速ではあるが小容量であり，大きいと低速かつ大容量である．PowerPC/7455系だと，L1＝64Kバイト，L2＝256Kバイト，L3＝2Mバイトのようになる．

PS/2デバイス

IBM社の元祖PC/ATパソコンで規定された，2線式シリアルによりデータ転送を行うデバイス．キーボードやマウスが代表的なPS/2デバイスである．

QVGA

320×240の画素表示領域をもつ画面解像度．VGAに対して，1/4の表示面積ということで，Quarter-VGA：クォータVGAと名づけられた．

RAMBUS

米国ラムバス社が提唱するラムバスメモリの総称．メモリだけでなく，クロック生成回路や信号インターフェース，基板設計手法など細かいところまで規格化しているところもラムバス社の特徴である．

RAStoCAS時間

同期式DRAMを使用した際，RASアクセスによるバンクの活性化(アクティブ化)からCASアクセスによる最終アドレス決定までの待ち時間．通常「t_{RCD}」というパラメータで表される．

RLDRAM (Reduce Latency DRAM)

Infinion社とMicron社で共同開発されたDDRメモリを改良したもので，従来のDRAMでは苦手だったランダムアクセス時におけるレイテンシを短縮したメモリ．

SCSI (Small Computer System Interface)

基本的には8ビットデータバスで，最大8台までのデバイスが接続できるストレージ規格．HDD以外に，CD-ROMやイメージスキャナなどにも使われている．バスの制御権を取得したデバイスがバスを使い，転送が終わればバスを開放するという，それぞれのデバイスが対等な関係となる．時代の流れとともにSCSI-1，SCSI-2，SCSI-3というように規格化され，高速転送への対応や，データバス幅の広いWIDE SCSI，また伝送距離を伸ばすためのディファレンシャル伝送などの規格が盛り込まれている．

SDRAM

クロックの両エッジでデータアクセスを行えるDDR-SDRAMに対して，クロックの立上りエッジのみで駆動する「シングルエッジ-SDRAM」の略称．

SH-2

日立製作所社製の小～中規模組み込み系プロセッサ．32ビットRISC形式で，内蔵メモリやタイマ，DMAの個数などで非常に多くの派生品がある．

SH-3

日立製作所社製の高性能32ビットRISCプロセッサファミリ．SH-2に比べて信号処理機能や演算性能の大幅な向上が施されている．携帯電話でJavaアプリケーションのデコーダとしても利用されている．

SH-4

SH-3のさらなる機能向上を施し，かつ動作周波数あたりの消費電力を大幅に減らした組

み込み機器用の高性能32ビットRISCプロセッサ．画像処理系に有効なベクトル演算ユニットを標準で搭載し，SDRAMメモリやPCIバスコントローラを搭載した派生品もある．

SIMM (Single Inline Memory Module)

8ビットや32ビットデータバスをもつメモリモジュール．DIMMにくらべて搭載できるメモリ容量が少ないため，現在はDIMMが主流である．

SIO (Serial Input Output)

シリアル入出力を行う機能デバイスのことを示す．UARTとも呼ぶ．

SO-DIMM

JEDEC規格で規定された，小型形状のDIMMメモリ形状規格．DIMMの約半分の大きさでDIMMと同一の信号線が包括されている．主としてノートパソコン用に用いられる．

Spartan II

米国ザイリンクス社から提供されるFPGAデバイスのファミリ群総称．従来のFPGAに比べてDLLやメモリ機能の内蔵などで高機能なうえに，低コストのために人気がある．

SPD

DIMMメモリ固有の，メモリ容量，バンク数，RAS/CASアドレス幅などの個別データを保存したROM素子．I²Cシリアルで送受信を行うため「Serial Present Detect」と呼ばれる．

SPLD (Simple PLD)

PALやGALなどの比較的小規模なPLDデバイスを示す．

SSTL-1.8/2/3

SSTLはStub Series Termination Logicの略で，後の数字は駆動電圧を示す．JEDECにより規格化された信号伝送方式であり，小振幅の電位で300MHz程度までの信号伝送を行うことが可能である．DDRメモリのインターフェースとして一般的である．

SuperI/O

PC/ATアーキテクチャで，比較的低速なUSB，パラレルポート，タイマ，UART(非同期シリアル)，PS/2インターフェースなどを1チップに収めたデバイスの総称．

SVGA

800 × 600の画素表示領域をもつ画面解像度．

SXGA

1280 × 1024の画素表示領域をもつ画面解像度．

TA*

Transfer Acknowledge信号の省略名称で，PowerPC系におけるデータ転送の完了通知を示す．MC68000/020/030やVMEではDTACK*と同義の信号だが，こちらはクロック同期．

TEA*

Transfer Error Acknowledge信号の略で，PowerPC系におけるデータ転送時にエラーが発生して強制終了したことを示す．M68KではBERR*：バスエラー信号が類義である．TA*同様，クロック同期の信号．

TrueIDE

PCカードやCompactFlashカードのATAカードの動作モードの一つで，信号がIDEと完全に互換性のある動作をする動作モード．カード側はOE#信号に“L”レベルを入力しながら電源を入れるとTrueIDEモードになる．

TS*

PowerPC系のローカルバスインターフェースで，データ転送の開始を示す負論理の信号．クロック同期で1クロック期間のみアサートされ，アドレスやデータの確定を示すことにも使われる．

TTL

主としてバイポーラトランジスタで構成された論理ICの種類で，CMOSと対を成して製造プロセスの区別を表す．論理ICの代表格である74xxファミリはTTLプロセスである．

TTLレベル

5V駆動のTTLデバイスを使用する際のインターフェースレベルを表す．一般的に出力“H”レベル＝2.0V(最小)，出力“L”レベル＝0.8V(最大)という電位．

UART (Universal Asynchronous Receiver/Transmitter)

とくにシリアル通信デバイスのことをUARTやSIOと呼ぶ．MotorolaのMC2681は1チャネルのシリアル通信が行えるのでUARTだが，2チャネル内蔵のMC68681はDual-UART：略してDUARTと呼ぶ．

UDMA転送

UDMAコントローラによって行われるデータ転送．UDMAを使うことで66Mバイト/秒(33MHz駆動)や133Mバイト/秒(66MHz駆動)などが行える．

UltraDMA/UDMA

IDEインターフェースをもつHDD系のデータ転送時に，CPUによるPIO転送になりかわって行うDMAコントローラの名称．

UMA (Unified Memory Architecture)

プログラムを格納するメインメモリとビデオメモリを共有化して，システム全体のコストを下げた方式．

Verilog-HDL

米ケイデンス社が論理回路のシミュレーション用に開発した，C言語に似たIC開発言語．アナログ設計も包括したVerilog-AMSや，システム設計系をも包括したSystem-Verilogなどへの進化が行われている．

VESA規格

VESAグループが制定したビデオ関連の規格の総称．VGA，XGA，SXGAやUXGAなどの言葉と画面解像度，動作周波数などを含むありとあらゆるビデオ系の規格がこまかく規定されている．

VGA

640 × 480の画素表示領域をもつ画面解像度．

VHDL

米国防総省のVHSIC開発を目的としたIC開発言語．自社ツール向けのVerilog-HDLとは違い，安価なFPGA開発ツールの登場で一般的に広まった．Verilog-HDLに比べて厳格さが要求される言語形態であるため，記述量が多いのが難点という声も聞かれる．

VMEバス

IEEE1014-1987で制定された，工業計測機器や量子力学系研究機関システムなどで広く使われているシステムバス規格．M68Kバスと酷似したバスアーキテクチャである．シングルマスタ，優先順位付割り込み，64ビット

バス幅，DMA機能の標準サポートなどで，現在も何ら遜色のない能力をもつ．

VRAM

ビデオ回路系で画面表示を行うデータを格納するメモリの総称．Video-RAMと呼ばれるが，物理的な構成はSDRAMやDDR-SDRAMで行われるのが一般的である．

V-Sync同期

垂直帰線信号に同期したタイミングで画像表示を更新する方式．V-SYNC同期のビデオはティアリング（ちらつき）などが発生しない．一般にV-Sync同期を実現するためにはフリップ動作が必要である．通常のWindowsは非V-Sync同期であり，これが一般的になってしまったため，「小汚い」ビデオ出力で満足されるのが，非常に悲しい．

WAVファイル

Windowsシステムの標準オーディオ記録フォーマット．量子化数8，16ビットや22k，44.1kなどのサンプリングレートによるPCMデータ形式が一般的である．

XGA

1024×768画素サイズの画像解像度．

x86

Intel社8086，80186，80286……からPentiumファミリ（586アーキテクチャ）まで一貫した思想の16/32ビットマイクロプロセッサ群アーキテクチャの総称．

Zバッファ

複数の画面表示物体が存在するとき，奥行き概念を保存しておくバッファ．縦横のX/Yに対して，奥行きであるためZが充てられる．3次元表示系ではZバッファを代表する奥行きを管理する手段がないと，重ね合わせを行った際に不完全になってしまう．

信号名#

信号名にシャープ（#）を付加することで，信号が負論理であることを表記する．Intel表記やPCIバス規格の表記として使われる．

*信号名

信号の前にアスタリスク（*）をつけることで信号のレベルが負論理であることを示す表記方法．モトローラ表記では一般的である．

五十音

アーキテクチャ

CPU，グラフィックス，数値演算や信号処理などで，技術者がその構築/実現方法/設計概念や基礎理論を示す場合に用いられる．システム構築の根幹をなすものを説明する場合にも用いられる．

アービタ

一つのリソースに対する複数のアクセス要求があった際に，ある瞬間には1リソースに対して1アクセス要求しか受け付けないように，排他的処理を行う調停機能．

アービトレーション

アービタ機能により調停される「動作そのもの」を示す．PCIバスアービタによるバスマスタ/アービトレーションなどの使われ方がなされる．

アイドルサイクル

CPUやPCIのバスサイクルのうち，何も外部アクセスが行われていない，バスの未使用期間のことを示す．アイドルサイクルであれば，バスマスタのだれもがバスを使用することができる．

アクセスウェイト

PCIバスアクセスにおいて，データ転送が完了になるまでバスを占有したまま待ち状態になることを示す．アクセスウェイトが大きいと，パフォーマンスが悪くなる．

アクセラレーション

アクセラレータにより性能を向上させること．CPUでは処理負荷が大きい3次元画像処理系を，グラフィックスエンジンを追加して性能を向上させる行為そのものを示す．

アクセラレータ

ハードやソフトを追加して，システムの性能を大幅に向上させる機能．MacやX68000向けに，低能力のCPUを最新のCPUに置き換えて外見そのままに性能を向上させるCPUアクセラレータなどが有名である．

アサート/ディアサート

Intel表記で，信号が有効状態，または無効状態を示す．このほか，PCIバス規格の信号状態も「アサート/ディアサート」で表記される．

アサート/ネゲート

Motorola表記で，信号が有効状態，または無効状態を示す．Intel表記と対を成す．

インタリーブ

メモリなどへの読み出しアクセスが行われた際，一定のデータ転送量以上のデータを自動的に継続して読み出すことで「見かけ上の性能を上げる」方式．

インタラプト

現在進行中の処理状況に対して，外部から割り込んで処理要求を行う方式．割り込み要求．要求された内容が現在の処理系よりも高位であればシステムは割り込みを受け付ける．

インタラプトコントローラ

割り込み処理機構．要求された内容が現在の処理系と比較して高位であるか低位であるかを判定し，高位であれば現在の状態を保存して処理の切り替えなどを行う．

ウェイト

一連のデータ転送（またはトランザクション）の完了までの待ち状態を示す．PCIバスの場合，データフェーズに移行してからターゲットデバイスからのターミネーション応答が行われるまではウェイト状態である．

エージェント

PCIバス上におけるデータ転送中の「バスマスタデバイス」と「ターゲットデバイス」をまとめた呼称．PCIバス規格では1クロックごとに1組のエージェントしか存在しない．

エリアxx

SH系プロセッサはIC内部にバスコントローラを包括しており，そのバスコントローラが外部バスをアクセスする空間を「エリア」という概念で区別する．SH-4の場合，外部に七つのエリアをもつ．

オートプリチャージ

SDRAMを使用する場合，異なるRASやバンクをアクセスする場合には現在使用中のバンクをプリチャージして使用の完了を通知しなければならない．オートプリチャージは，メモリアクセス完了後に自動的にプリチャージを行うコマンドである．

オープンコレクタ
出力トランジスタのコレクタがオープンになっており，外部抵抗によってプルアップドライブを行う出力方式．リセット系や割り込み系/システムエラー検出系（PCI）などのように，複数信号のワイヤードオアができる特徴がある．

オープンドレイン
出力トランジスタのドレインがオープンになっており，外部抵抗によってプルアップドライブを行う出力方式．オープンコレクタと同一だが，こちらはMOS-FETを意識した言葉．

仮想メモリ
HDDの領域を物理メモリ空間としてアサインしたり，物理メモリの一部とHDDの中身を入れ替えて，実際には存在しないメモリ空間が，あたかも存在するように見せかけるテクニック．OSとMMUにより実現可能な，高度なハードウェアリソース管理手法．

キャッシュメモリ
メインメモリなどから読み出したデータを再利用するために，アドレス情報などのタグ情報をつけて一時的に保存しておくメモリ．メインメモリなどよりもはるかに高速で動く．

コンフィグレーションサイクル
PCIバスにおける転送方式の一つであり，通常PCIデバイスの初期化フェーズで発行されるアクセス方式．本アクセスにより，ベースアドレスや割り込みなどのシステムリソースが割り当てられる．

システムエラー
PCIバス上で異常を検出した際の状態の一つ．通常はアドレスフェーズ中にパリティエラーが発生すると即時にシステムエラーとして検出される．SERR#信号のアサートを促す．パリティエラーはこの後に続くデータフェーズで検出されるものである．

システムロック
システムが何も応答しない状態．PCIバスシステムの設計時にシステムロックになった場合は，TRDY#応答が正常に行われていない場合がほとんどである．

時分割バス
マルチプレクスバスの日本語名．時間ごとにバス上におかれているデータの意味が変わる．通常はバスクロックに応じてバス内のデータが切り替わる．MPXバスと同義語．

シリアルATA
ATA/ATAPIなどIDEは，16ビットデータバス幅で高速化してきたが，パラレルのままではクロックに対するデータのスキューが問題となり，高速化には限界がある．そこでデータ転送をシリアルにしてHDDやCD-ROMドライブを接続するための規格がシリアルATAである．

スタック
積み重ねるという状態を表す言葉で，データを積み重ねて利用する機構．積み重ねるために後から置いたものを先に取り出す．一般的にはサブルーチンコール時の状態切り替え時に内部レジスタなどを退避する方式として，スタック方式が用いられる．

スプライト
任意の矩形領域をひとまとめの画像出力データとして管理し，画面に表示する方法．2D系システムではマウスカーソルやキープロンプトなどに利用される．

スリープ動作
SH-4などの組み込み系マイコンや低消費電力システムにおいて，処理系を一時的に停止して消費電力を低くした状態を示す．再度のキー入力などで再度通常動作状態に復帰する．

スリープモード
スリープ動作後の動作停止状態．SH-4の場合には内蔵PLLやSDRAMメモリへの供給クロックを停止するなどのスリープ動作を行った後にスリープモードへ移行する．

セカンダリバス
PCItoPCIバスブリッジにおいて，ホストデバイス側から遠いところに位置するバスの呼称．低速なI/Oデバイスなどが接続される．

センスアンプ
メモリIC内部で，ロウアドレスとカラムアドレスから読み出された極小のビットデータを増幅して出力バッファに送り出す増幅回路．メモリセルの次に重要なDRAM構成部品の一つ．

ターゲットアボート
PCIバス上のマスタデバイスからのすべてのデータ転送要求に応答できないということで，ターゲットデバイスがデータ転送の打ち切りを要求する応答方式．

ターミネーション
PCIバス規格におけるデータ転送の完了方式の一つであり，日本語でいう「打ち切り」という強い意味ではなく，正常/異常を含む一連のデータ転送の完了動作を示す．

ダブルバッファリング
高速なデバイスと低速なデバイスとの間でデータ転送を円滑に行うため，二つのバッファを用意して交互にデータの書き込み/読み出しを行う方式．画像系ではとくに有効．

チップセット
システムを構成するために，CPU以外のメモリコントローラやPCIホストブリッジ，ビデオ，HDDコントローラなどの各種I/O系のデバイスの組み合わせを示す．

調歩同期
非同期に送られてくるデータを基準クロックに同期して信号変換を行う通信方式．1対の送受信信号線で通信されるRS-232-Cシリアル通信は，調歩同期式シリアル通信である．

ティアリング
画像出力系の用語．動画の「コマ落ち」や「ちらつき」が発生している状態を表す．

ディスコネクト
PCIバス上のマスタデバイスからのバースト転送に応答できないということで，ターゲットデバイスによるSTOP#アサートによるデータ転送の中断要求状態を示す．

テクスチャ
「質感」を示す画像出力系の言葉であり，たとえば金属や石，木などの質感データを用いてコンピュータで処理した画像の品質を向上させるために利用されるデータを示す．

バースト転送
データ転送系において，連続したデータ転送を行う場合の名称．4バーストというと，4データ連続アクセスを示す．

ハイカラー
画像系で，RGB-16bpp・65,536色の表現ができるカラー映像のことを示す．Windowsパソ

コンでは「16ビットフルカラーモード」ともいう.

ハイレゾ
(High Resolution:高解像度の略)

昔は640×480(VGA)を超える解像度をもつ画像出力系に対して呼ばれた.現在ではXGAは一般的なのでSXGA以上の解像度をいう.

バスアービタ

アービタ機能を包括して,バスシステムに特化したアービトレーション制御回路のこと.PCIバスの場合には各バスごとに1個のバスアービタが存在する.

バスアービトレーション

PCIバスシステム上で,バスの使用権をバスアービタに要求し,バスアービタがバスの使用権を譲渡する一連の動作を示す.REQ#/GNT#の2線で行われる.

バスパーキング

PCIバスシステム上で,前回バスを使用したバスマスタデバイスに,要求がなくとも引き続きバス使用権利を与える機構.通常は連続してバスを使用するであろうとの予測に基づく.

バスマスタ

バスアービタによってバス使用権利が与えられており,PCIバス上にデータの入出力を行うことができるPCIデバイスの呼称.

バスマスタデバイス

バス使用権利をもつPCIデバイスであり,1バス上には1クロックあたり一つのバスマスタのみ存在する.

バスマスタ転送

PCIのターゲットデバイスからのデータ読み出しは書き込みに対して遅いため,そのためターゲットデバイス内のDMACを駆動してマスタデバイスに切り替え,書き込み転送を行ってもらうデータ転送方式.CPUやDMACのサポートが必須.

バックプレーン

VMEバスやFastBUSなどの,ボードを複数枚接続して駆動するための電源供給/バス信号引き回しを行う基板.カードエッジPCIの場合はマザーボードというが,CompactPCIの場合にはバックプレーンという.ラックマウントするかどうかで使い分けている.

パリティ

データの信頼性を確保するための機能の一つ.あるまとまったデータ群の1の偶数/奇数を求め,偶奇性がゼロになるような方向で値を決める.1ビット分のエラー検出は行えるが,エラー訂正や2ビット以上のエラー検出は行えない.

パリティエラー

PCIバス上で異常を検出した際の状態の一つ.通常はデータフェーズ中にパリティエラーが発生すると即時にパリティエラーとして検出される.PERR#信号のアサートを促す.システムエラーよりは緊急度は低いが,それをどう使うかはシステムのアーキテクチャに依存する.

ビートアクセス

バースト転送と類義語で,PowerPC系の場合にとくにキャッシュへの充填転送の際に使われる言葉.G4系では64ビットのデータを1ビートといい,1,2,4ビートアクセスが自動的に行われる.

ピクセルレート

CRTやLCDに表示される画面は一つ一つの画素データから成り立つ.その画素データの入出力を行う際の速度を「Pixel-Rate:ピクセルレート」と呼ぶ.また画素レートともいう.

フェイドコントロール

画面が徐々に表示されることをフェイドイン,反対に徐々に表示されなくなることをフェイドアウトといい,透明度演算回路(アルファブレンド機構)により実現される制御回路.

プライマリバス

PCItoPCIバスブリッジにおいて,ホストデバイス側に位置するPCIバスの呼称.グラフィックスやSCSI-HDDなどの高速なI/Oデバイスなどが接続される.反対側にはセカンダリバスが存在する.

ブランク

ビデオ回路系における「無表示」,「空白」を表す.表示系では,上下左右の各端に一定のブランク期間を設けておくことで,画面がゆがんだりVRAMの非表示画素データが表示されなくなる.

プリチャージ

同期式DRAMを使用する場合において,異なるRASやバンクをアクセスする場合に,現在使用中のバンクの完了通知の動作を行う際の言葉.

フリップ

ダブルバッファリング方式を採用した画像出力系において,画面を切り替えることを示す.フリップタイミングをV-SYNC同期で行うことで,ちらつきが発生せず,動画系には最適である.

ブルーバック画面

PCIバスシステムのデバッグ中に,設計回路が誤ってアドレスフェーズやデータフェーズで応答してしまった際に陥るWindowsのアボート現象.システムロックと同様にPCIバスの不具合が発生していることを知る有効な(!?)デバッグ画面である.

フルカラー

画像系で,RGB-24bpp・約1677万色の表現ができるカラー映像のことを示す.Windowsパソコンでは「24ビットフルカラーモード」ともいう.

フルページバースト

SDRAMメモリの特殊な使い方で,通常2,4,8程度のバースト転送以外に,CASアドレス範囲すべてを一気に読み出すことのできる特殊なバースト転送方式.リニアバーストとも呼ばれる.

フレームバッファ

CRTやLCDに表示される画面のことをフレームといい,表示するフレームデータを格納してある一時記憶領域をフレームバッファと呼ぶ.フレームバッファの内容は常時書き換わり,ダブルバッファリングとフリップにより,なめらかな動画再生やちらつきのない表示を行うことができる.

フレームメモリ

VRAMと同義.画面表示を行う際に表示を行う1画面(=フレームと称する)分以上のフレームデータを格納するメモリのことを示す.

ベースアドレス

PCIバス上のデバイスがシステムやOS上の特定の空間に割り当てられた基準アドレス.PCIデバイスには最低1個のベースアドレス

Information ── PalmがHandspringを買収
米Palmと米Handspringの両取締役会は,Palm社がHandspring社を買収することについて合意した.また,Palm OS事業を手がける同社傘下の米PalmSourceは完全に独立する.

レジスタが実装されており，そこにアドレスが設定されている．

ベクタスキャン

CRTを利用し，直接走査線を磁場で操作して画面に出力する方式．アナログオシロスコープが一般的だが，LCDに変わりつつある現在は，使用される場面が限られている．現在はほとんどの表示系がラスタスキャン方式である．

ベクタ割り込み

割り込み要求時に「割り込み処理プログラムの番地やポインタ」情報を付加して割り込み要求を行う手法．割り込みレベルと併用して高速に割り込み処理プログラムを起動できるという利点がある．

ホストバス

PCIを採用した機器のうち，ホストブリッジ直下の通常バス番号0になるPCIバス．下位に接続されるPCIデバイスの円滑なデータ転送のため，高速なPCI-66/PCI-Xバスなどの採用が最適である．

ホストブリッジ

CPUとPCIバスの間に位置し，CPUからのアクセス要求によりPCIバス側にアクセスを行う統合チップ．CPUバスの複雑なバスインターフェースをPCIに切り替える機能をもっており，ブリッジ＝橋渡しという意味合いがある．原則として1-PCIシステムでは一つのホストブリッジが存在する．

マスタアボート

PCIバスのデータ転送の際に，バスマスタデバイスがデータ転送をキャンセルした，ターミネーション方式．アクセス先のターゲットデバイスが応答しない場合に発行される．

マッピング

「貼り付ける」ということを示す画像出力系の言葉であり，テクスチャ（質感）を貼り付けた場合はテクスチャマッピング，でこぼこ感を表すことをバンプマッピングという．CPU処理では膨大な処理量になるため，3Dアクセラレータの得意とする処理となる．

マルチバスマスタシステム

一つのバスに複数のバスマスタデバイスが存在するシステムの呼称．とくにPCIバスシステムの場合には，すべてのPCIデバイスがバスマスタになり得える．このため，必ずバスアービタが搭載される．

マルチプレクスバス

一つのバスを時分割で複数の信号が使うバス方式．PCIバスの場合にはアドレスとデータが同一のバスを使用する．SDRAM系ではアドレスバスをRAS/CASで利用する．

メモリバンク

SDRAMやDDR-SDRAMを構成する記憶領域＝メモリのまとまった容量のかたまりをバンクという．メモリアクセスの際に活性化される（アクティブ化）される記憶領域を呼ぶ場合に，この言葉が使われる．

ラスタスキャン

CRTやLCDへの画面表示方法の一つで，水平同期信号（H-SYNC）と垂直同期信号（V-SYNC）で一つの走査線を横方向（ラスタ）にならべて1画面を生成する方式．

ラップアラウンド

定められたデータ転送長を超えて，データの最終まで読むと自動的にデータの先頭に戻って継続してデータアクセスを行う動作．キャッシュアクセスが有効であると，頻繁に行われる．

ラドハード（Radiation Hardness：ラジエーションハードネスの短縮語）

放射線に対して高い防護能力をもつICの種類を示す．MIL-883C/Dよりもさらに厳しい試験が行われる．また，一般的に普通の半導体メーカーではラドハードに対応しておらず，米モトローラ社・MC68000のラドハード品がトムソン社・TS68000というように，国防産業に強いサードベンダがライセンス契約を行って製造を担当する．

リトライ

現在のマスタデバイスからのデータ転送要求に対して，ターゲットデバイスがすぐには応答できず，いったんデータ転送を打ち切り再度試行してもらうことを要求する応答方式．

リフレッシュ

SDRAMやDDR-SDRAMのようなものは記憶部位がコンデンサで構成されているため，一定期間ごとに再活性化を行わねばならない．この行動をリフレッシュと呼ぶ．

リフレッシュアービタ

リフレッシュ動作を行う際に，CPUやDMAなどのほかにメモリを使用するハードウェアリソースとのアクセス調停を行う機能．通常メモリコントローラに内蔵されている．

リフレッシュレート

メモリ系の場合，リフレッシュを行う頻度を示す．データシートなどで4096回/64msのように表記されており，この場合，15.6μsごとに1回のリフレッシュを行う．ビデオ回路系の場合は，画面の表示更新頻度を示す．一般的なモニタでは1秒間に60回以上の表示更新を行っており，更新頻度が低いと，「ちらつき」をおこす．

ローカルバス

PCIバスやISAバスなどの他のバス規格に対して，CPU自身の元々のバスの呼び名．通常はボード上のどのバスよりもローカルバスがいちばん速い．

ロータリエンコーダ

ツマミやネジまわしがついたスイッチの集合体であり，通常はツマミ位置に応じた4ビットのデータが出力される．ボードに個別IDを振り分けるような場合に広く用いられる．

井倉将実　来栖川電工有限会社

chapter 08

無線による高速データ伝送が身近になってきた！

ワイヤレスネットワーク技術入門

本章では，さまざまな規格が標準化され，規格に準拠した機器も多種多様に登場しているワイヤレスネットワーク技術に関する用語を解説する．本誌2003年2月号の特集「ワイヤレスネットワーク技術入門」では，現在もっとも普及しているIEEE802.11方式の無線LAN技術の基礎や標準化動向，Linux上で動作するBluetoothプロトコルスタックとその評価ボード，IEEE802.11aで変調方式として使われるOFDMの基礎原理およびOFDMデータモデムの設計事例，100Mbps以上のデータ伝送を実現する，60GHz帯の電波を使った無線伝送技術の基礎と「ミリ波自己ヘテロダイン伝送方式」，同じく100Mbps以上の伝送速度を実現する通信方式「UWB」の基礎から技術的課題・今後の発展性などをくわしく解説した．

ワイヤレスネットワーク技術は，今後ますます激しい展開が予想されるが，この分野特有の言い回しや用語も多い．今回の用語解説を一読したうえで，前記の特集も再読し，理解を深めていただければ幸いである．

（編集部）

ワイヤレスネットワーク技術の現況

16QAM（16-position Quadrature Amplitude Modulation）

16値直交振幅変調．データに応じて搬送波の位相，振幅を16種の組み合わせに切り替える．1変調シンボルあたり情報を4ビット伝送できる．

2進指数バックオフアルゴリズム

フレームの衝突が起こり，再送するたびにバックオフ時間を決める乱数発生の範囲を2倍に増加させていくバックオフ制御法．これにより，再衝突確率を低減させる．

64QAM（64-position Quadrature Amplitude Modulation）

64値直交振幅変調．データに応じて搬送波の位相，振幅を64種の組み合わせに切り替える．1変調シンボルあたり情報を6ビット伝送できる．

BPSK（Binary Phase Shift Keying）

2相位相変調．データに応じて搬送波の位相を0°と180°に切り替える．1変調シンボルあたり情報を1ビット伝送できる．

CCK-OFDM方式

IEEE802.11gの標準化に提案されたIEEE802.11bとaの融合方式．現在は，「DSSS-OFDM方式」と呼ばれている．

CCK（Complementary Code Keying）**方式**

相補型符号変調方式．IEEE802.11bに必須の方式として規定されている．直接スペクトル拡散方式の一種で，拡散符号にComplementary Codeを用い，情報要素をもたせることで高速化をはかっている．CCK方式とDS-SS方式について，**図1**（次頁）に示す．

→DS-SS方式

CSMA/CA（Carrier Sense Multiple Access with Collision Avoidance）

他局の信号に衝突させないように，事前にチャネルの使用状況を確認（キャリアセンス）し，送信タイミングを自律的に制御する方式．**図2**（p.97）に，CSMA/CAによるアクセス制御を示す．

DCF（Distributed Coordination Function）

IEEE802.11で定義されるCSMA/CAをベースとした自律分散的な媒体アクセス制御方式によりデータを伝送する手順．

DIFS（DCF Inter Frame Space）

分散制御用フレーム間隔．IEEE802.11規定のDCFにおいて，キャリアセンスを行う際，チャネルが使用中から空き状態に変化したと判断されるのに必要なチャネルの未使用時間．

DSSS-OFDM方式

IEEE802.11gに提案されたIEEE802.11bとaの融合方式．IEEE802.11bと互換性を保つため，ヘッダ部までをDSSS方式，データ部を高速伝送するためOFDM方式としている．ヘッダ部までが長いため，実効スループットが向上しない．

DS-SS（Direct Sequence Spread Spectrum）**方式**

直接シーケンススペクトル拡散．IEEE802.11の物理レイヤ方式の一つで，11bでも必須の方式となっている．1MbpsのBPSKまたは，2MbpsのQPSKを11chipのBarker Codeで拡散する．DS-SS方式とCCK方式について，**図1**に示した．

→CCK方式

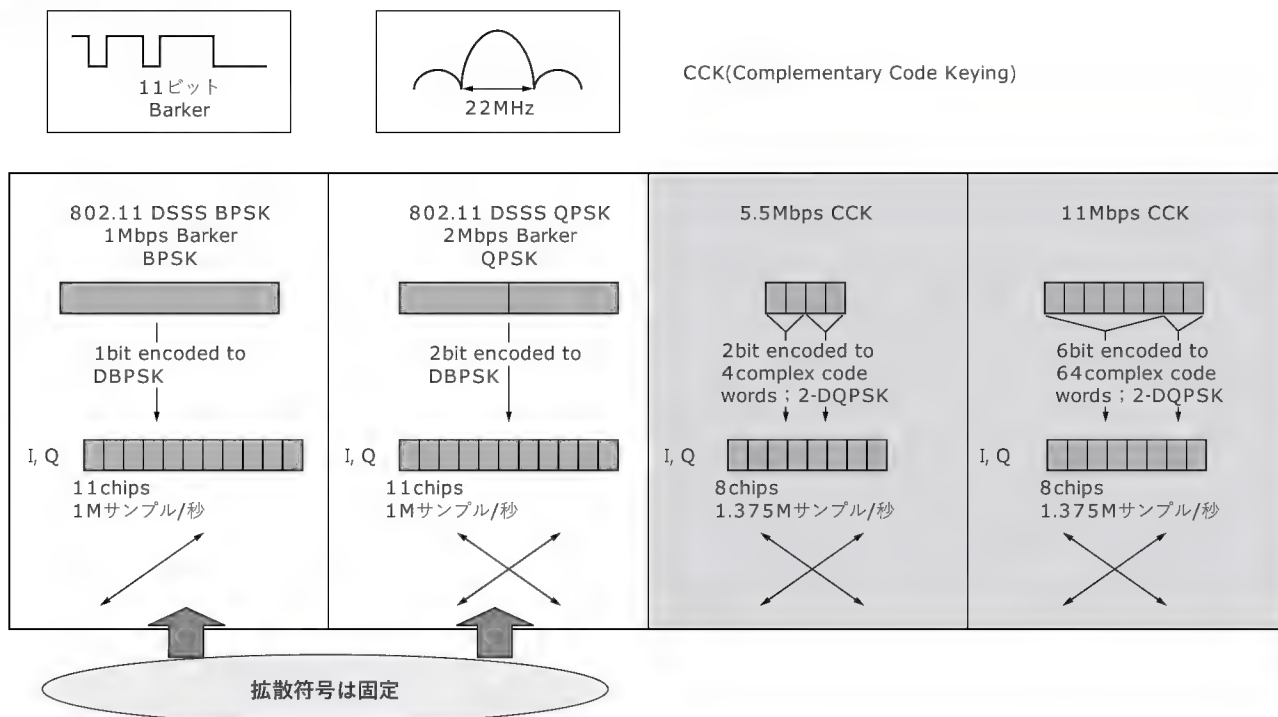
EAP-TLS（Extensible Authentication Protocol-Transport Layer Security）

さまざまな認証方法をサポートするプロトコルであるEAPの上で，公開鍵認証方式の一種であるTLSによって認証する方法．Windows XPに標準サポートされている．

EAPOL（Extensible Authentication Protocol over LAN）

LAN上に動作する拡張可能な認証プロトコル．IEEE802.1Xに規定されているEAPのメッセージをLAN上で伝送するためのしくみ．

〔図1〕CCK方式とDS-SS方式

CCK(Complementary Code Keying)

HCF競合チャネルアクセス

IEEE802.11eで検討されている優先制御手順．従来のDCFを拡張したという意味で，「EDCF」ともいわれる．送信データを4種類のアクセスカテゴリに分類し，カテゴリごとに提供するサービス品質に差を付ける優先制御を提供する．

HCFポールドアクセス

IEEE802.11eで検討されている帯域保証手順で，ポーリングを用いた伝送方式であり，従来のPCFを拡張した手順である．ポーリングする際に所要品質を考慮したスケジューリングを行うことで，指定帯域幅や遅延時間などを保証するQoSをサポートできる．

IEEE802.11

無線LANの標準規格．1997年に制定され，2.4GHz帯の電波および赤外線を用いた1Mbpsおよび2Mbpsの物理レイヤと，MAC(媒体アクセス制御)レイヤの規定からなる．

IEEE802.11a

1999年に制定された5GHz帯の電波を用いる無線LANの標準規格．変調方式に，マルチパス干渉に強いOFDMを適用し，6～54Mbpsの伝送速度が規定されている．

IEEE802.11b

IEEE802.11の2.4GHz帯の電波を用いる直接拡散方式をもとに，伝送速度の高速化をはかった無線LAN標準規格．IEEE802.11に対し，5.5Mbpsおよび11Mbpsの速度が追加された．

IEEE802.11ワーキンググループ

米国IEEE(Institute of Electrical and Electronics Engineers)の802委員会配下にあるワーキンググループ．1990年に設立された無線LANの標準化を行う代表的機関である．

IFS(Inter Frame Space)

IEEE802.11で定義されている送出信号の間隔(フレーム間隔)．チャネルが使用中から空き状態への移行を契機にIFS時間だけ待つ．IFSの長さにより無線局間の優先度をつけることができる．

ISDN(Integrated Services Digital Network)

音声やデータを統合して扱うディジタル通信網で，一般に制御用Dチャネル(16kbps)，通信用Bチャネル(64kbps)の2本からなる．

MAC(Medium Access Control)レイヤ

MACは「媒体アクセス制御」を意味し，どのタイミングでネットワーク媒体(有線ではケーブル，無線では空間)に信号を送出するかを制御する方式(プロトコル)レイヤである．

OFDM(Orthogonal Frequency Division Multiplexing)

直交周波数分割多重．IEEE802.11a/gの変調方式として規定されている．信号を複数のサブキャリアに分割して伝送するため，多重波干渉による歪みの影響を受けにくいという特徴をもつ．

PBCC方式(Packet Binary Convolutional Coding)

パケット型2値畳み込み符号化のこと．IEEE802.11bおよびgに提案された．基本的には畳み込み符号とPSK変調を組み合わせた方式．

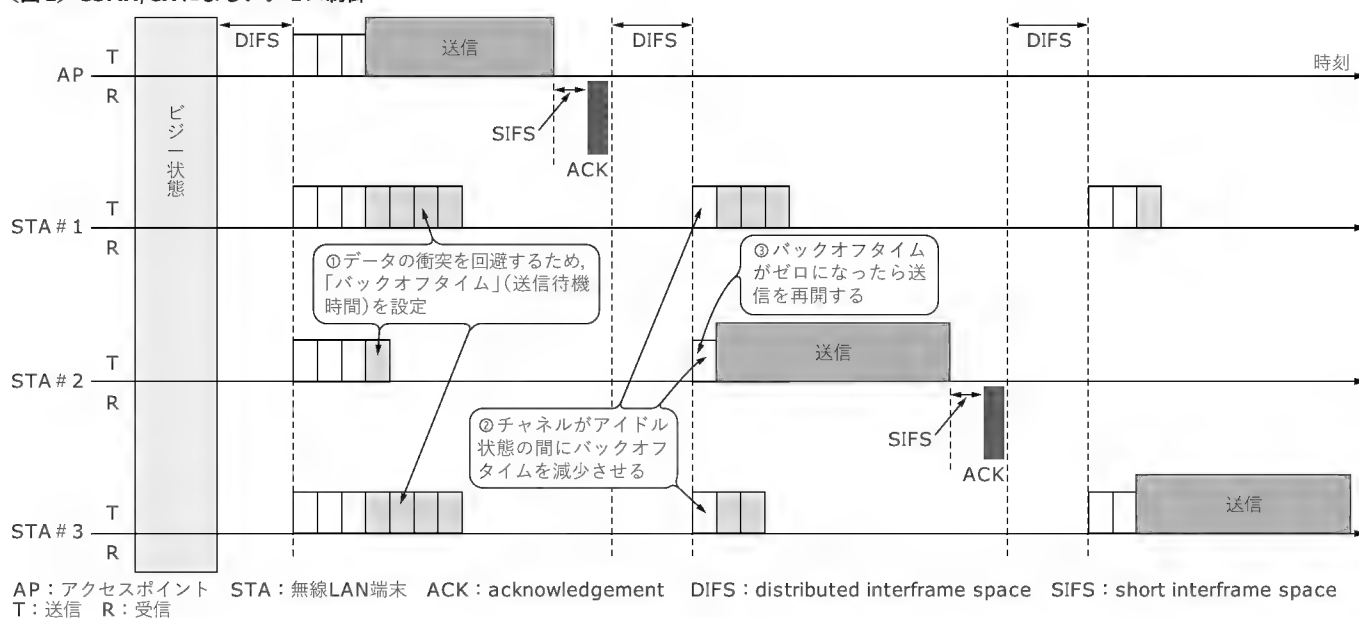
PCF(Point Coordination Function)

IEEE802.11でオプション規定のポーリングに基づく集中制御による媒体アクセス制御方法．一般にアクセスポイントが自配下の端末局を集中制御する．

PLCP(Physical Layer Convergence Protocol)副層

伝送速度やフレームの長さなど物理レイヤの

〔図2〕CSMA/CAによるアクセス制御

AP：アクセスポイント　STA：無線LAN端末　ACK：acknowledgement　DIFS：distributed interframe space　SIFS：short interframe space
T：送信　R：受信

情報をやり取りする通知プロトコル副層を指す．

PMD(Physical Medium Dependent)副層

物理層媒体依存部．物理レイヤのうち，伝送媒体に依存する下位副層を指す．

QPSK (Quadrature Phase Shift Keying)

4相位相変調．データに応じて搬送波の位相を0°，90°，180°，270°に切り替える．1変調シンボルあたり情報を2ビット伝送できる．

RTS/CTS手順

Request to Send(送信要求)/Clear to Send(受信準備完了)．IEEE802.11で規定される隠れ端末問題対策．送受信局が通常のデータ送信の前にあらかじめRTS/CTS信号をやり取りすることで周囲に存在を知らせ，仮想的なキャリアセンスを行う．

TKIP (Temporal Key Integrity Protocol)

IEEE802.11iで検討されている暗号方式．WEPと同じRC4暗号を用いながら，周期的に暗号鍵を変更するなどで従来の問題点を解決している．すでにWPA(Wi-Fi Protected Access)として採用されている．

WEP(Wireless Equivalent Privacy)

IEEE802.11に規定された暗号方式．暗号化アルゴリズムにRC4(Rivest's Cipher 4)を用いる．事前に設定した固定鍵の使用，鍵長などの点からその安全性が問題視されている．

隠れ端末問題

無線通信で局の位置関係や障害物の影響で互いの電波が到達しない状態．キャリアセンスをベースにした媒体アクセス制御では，互いに存在を知り得ないため，フレーム衝突確率が増大し，問題となる．

基本サービスセット

BSS(Basic Service Set)．IEEE802.11で定義される無線LANシステムにおいて，基本となる一つの基地局とその配下にある複数の端末局から構成されるネットワーク．

バックオフアルゴリズム

キャリアセンスに加えて衝突回避し，送信機会を公平にするための方法．チャネルが空き状態になってIFS時間後，送信しようとする局は規定範囲内で乱数を発生させて決めたバックオフ時間分キャリアセンスを継続する．

物理レイヤ

データ伝送速度，無線周波数帯域，変調方式などのデータ信号の物理的な条件を規定するレイヤである．

ポーリング方式

制御の主管局(無線LANではアクセスポイント)が，各端末にデータ送信をしたいかどうかを問い合わせて，応答のあった端末に送信機会を与える集中制御型の媒体アクセス制御方式．

無線LAN

LANの標準的な伝送方式であるEthernetを配線なしで伝送する構内通信技術で，媒体に電波，赤外線，光を用いたものがあり，通常は，PCとLANを無線接続する形態で利用される．

無線LANアクセスポイント

無線LANシステムにおける基地局(親機)．バックボーンネットワークと接続されており，配下の端末局のネットワークへの通信を中継する．また，配下の端末局の管理機能を有している．

無線LAN端末

無線LANシステムにおける端末(子機)．一般には，アクセスポイントを介してネットワークとの通信を行う．ただし，端末同士で直接通信を行うアドホックネットワークモードもある．

レートアダプテーション機能

無線LANにおいて電波状況に応じて伝送速度を切り替える機能．電波状況が悪い場合には，伝送速度を低速モードへ切り替え(フォールバック)，逆に良い場合には高速モードへ切り替える．「リンクアダプテーショ

ン」，「マルチレート制御」とも呼ばれる．

ワイヤレスTA

通常，ケーブルで接続されるターミナルアダプタ（TA）とPC間を接続するRS-232-Cインターフェースを無線接続する機器．

Bluetoothプロトコルスタックの開発と検証

AVCTP〔Audio/Video Control Transfer Protocol（音声・映像伝送制御プロトコル）〕

音声・映像を送信するうえでの制御命令と応答メッセージのデータフォーマットを規定しているプロトコル．

AVDTP〔Audio/Video Distributed Transfer Protocol（音声・映像配信制御プロトコル）〕

音声・映像のストリーミング配信の際の各種制御手順（ネゴシエーション・経路確立・送信など）を規定しているプロトコル．

Bluetooth

スウェーデンのEricsson社が中心となり，策定された近距離無線通信規格．10～100m程度の通信距離をもち，データ転送速度は約1Mbps．ISMバンド（2.4GHz帯）を利用しており，世界各国共通で機器が使用できる．規格提唱の段階では，PCと携帯電話を接続する用途を目的としていたが，現在では，PC，携帯電話のみならず，プリンタ，モデム，カメラなどさまざまな機器に搭載されている．現在，IEEEにおいてBluetoothをベースとして（Bluetoothと互換がある）802.15.1（WPAN規格）の策定が行われている．

http://www.bluetooth.org/
http://www.ieee.org/

Bluetooth Core

Bluetoothプロトコルの根幹となる技術仕様書で，一般的に「コア」と呼ばれる．Bluetooth Core（Version 1.1）は，次のようなレイヤで構成される．

- RF（無線に関する電気的仕様）
- BaseBand（ベースバンド制御）
- LMP（リンク管理）
- L2CAP（論理リンク制御）
- SDP（サービス検出・管理）
- RFCOMM（RS-232-Cエミュレート）
- OBEX（オブジェクト交換）
- Telephony Control（テレフォニ制御）
- HCI（ホスト制御）

Bluetooth Profile

機器の特性に応じて（Bluetooth Coreで規定されているうちで）必要な各種プロトコル群が定義されており，Bluetooth機器間の相互接続を実現するための仕様書である．

Bluetooth SIG（Bluetooth Special Interest Group）

Bluetoothについて管理・運営を行う団体．Ericsson，Nokia，東芝，IBM，Intelの5社が中心となって創設された．Bluetooth SIGは「技術規格の標準化・提唱」や「Bluetooth対応製品の管理」などの活動を行っている．

BNEP（Bluetooth Network Encapcellaration Protocol）

Bluetooth機器で，ネットワーク機能を実現するためのプロトコル．IEEE802.3/Ethernetと同じ方法でデータのカプセル化を行っている．

CURSES

コンソールベースの画面制御ライブラリ．UNIX互換OS上でCURSESライブラリを使用することで，非機種依存のプログラム開発が容易に行える．

DUNP（Dial-Up Networking Profile）

Bluetooth機器で，ダイヤルアップ接続を行う機器向けのプロファイル．

IETF（Internet Engineering Task Force）

インターネット上で利用される各種プロトコルの標準化を行う団体．

http://www.ietf.org/

PAN（Personal Area Network）

1ユーザーを中心として，そのユーザーの周辺に存在し，ユーザー自身が使用するさまざまな機器や家電を接続した場合のネットワーク形態．

PPP（Point to Point Protocol）

各種通信機器を1対1で接続するためのプロトコルで，OSI参照モデルのデータリンク層に該当する．LCP（リンク制御）や，PPP Authentication Protocols（認証）などの機能が存在し，各機能はRFC（Request for Comments）で規定されている．

→PPxP

PPxP

真鍋敬上氏によりUNIX互換OS上へ実装された，PPPプロトコルを実現するためのソフトウェア．LinuxやFreeBSDなど各種のプラットホーム上で動作する．

http://www.linet.gr.jp/~manabe/PPxP/

→PPP

RFC（Request for Comments）

インターネットにおけるプロトコル技術，提案，改良などを記述した文書．IETF（Internet Engineering Task Force）にて公式に発表される．

→IETF

RFCOMM

Bluetooth Coreで規定されている，シリアルポート（RS-232-C）をソフトウェアエミュレーションするためのプロトコル．ETSI（European Telecommunications Standards Institute）で規定されている規格TS 07.10と互換がある．

UnPlugFest

Bluetooth機器の相互接続性テスト．世界各国の企業や団体がBluetooth対応の機器を持ち寄って，実際に接続テストを行い，接続の状態を検証して仕様部分のあいまいさや，実装の間違いを訂正し，相互接続性の向上をはかる試み．

キャラクタ型デバイス（キャラクタデバイス）

UNIXにおいて，データをキャラクタ単位で逐次入出力するタイプのデバイス．これに対して，データをある単位にまとめた後，入出力を行うデバイスを「ブロック型デバイス（ブロックデバイス）」と呼ぶ．

ピコネット

Bluetooth機器が構成する最小のネットワークの形態．1台のマスタ機に対して最大7台のスレーブ機が通信可能．

プロトコルスタック

ある通信規約にしたがった処理を行うソフト

ウェア．複数の機器やデバイスで通信を行うために使用される．

OFDM 無線モデムの基礎技術と設計事例

AGC

出力される信号レベルが常に一定になるように，前段に入っている可変利得増幅器を自動的に制御する回路，またはシステム．とくに受信機などでは，アンテナから拾う信号レベルのダイナミックレンジはとても広く，AGC を入れて常に復調できるレベルにしなければならない．

MATLAB

MathWorks 社のシミュレーションソフトの名称．システム設計の上流で，設計したシステムをさまざまな条件で事前にシュミレーションが行える．各応用分野に応じたあらかじめ組み込まれた処理系をまとめたパッケージがある．実物で実験しなくても，ある程度の評価が可能となる．

OFDM

直交した複数のキャリアをそれぞれ変調し，極限まで周波数多重した変調方式．信号処理上は高速フーリエ変換を使い変復調処理を行う．一つ一つのキャリアの変調速度は遅いため，フェージング条件に強い．地上波のディジタル TV 放送に採用され，注目されている．

PAR

信号のピークと平均値の比．QAM や OFDM 変調では，線形変調で PAR が大きいため，送信機の直線性が問題になる．ピークも歪ませないような送信機設計をすることは難しいので，大きな問題になる．そこで，PAR を下げる何らかの工夫が行われている．

SH-2

ルネサステクノロジ（日立製作所）の 32 ビット 1 チップ RISC CPU．ほとんどの信号処理を 1 命令のサイクルで実行できる．また内部に積和演算ハードウェアがあり，ディジタル信号処理も高速に処理できる．RAM，ROM をはじめ，ほとんどの周辺回路を集積している．

ガードインターバル

OFDM 変調で，マルチパスの影響を軽減するために必ず挿入される，データ通信とは直接関係ない時間領域．またこの信号部分の自己相関のピークを使い OFDM 通信における同期の基本位相を得ることができる．

キャリア

無線通信などで，変調の前の信号を乗せる高周波信号．無線機ではその高周波信号に振幅変調，位相変調，周波数変調などの処理を行い，必要な情報を伝達することができる．変調後キャリアの成分が残る場合もあるし，そうでない場合もある．

高速フーリエ変換

離散的フーリエ変換は，多くの畳み込み演算の集まりで，重い処理を必要とする．そこで実時間処理を可能とするため，それを高速処理できるように考えられたアルゴリズム．三角関数の周期性を利用して，重複演算するところを切り詰めたもの．

周波数オフセット

送信した信号と，受信した信号の周波数にズレが発生するときの，そのズレ量．原因としてはドップラー効果や，SSB 復調における同調ズレ，その他伝送路で変復調の際に発生する場合が多い．とくに OFDM 変調では深刻な影響を与える．
→ OFDM

周波数ホッピング

周波数拡散通信方式の一つ．一定時間ごとにキャリア周波数を広範囲に変化させる．変化パターンにランダムな符号化を行い，そのパターンがわからないかぎり，復調できない．また，マルチパスによる障害のような周波数依存性回線に適した変調方式．

シンボル同期

ディジタル変調の場合，データ伝送の基本単位であるシンボル単位にデータが送られる．そこで受信側では，正確にデータ列からシンボル間の区切りを見つけ，読み出さなければならない．このように，正確にシンボルを取り出す最適タイミングを取り出すこと．

シンボルレート

ディジタル変調における，変調の基本的単位．単位は Baud で，その速度を表す．1 シンボルで伝送できる情報量は変調方式による．たとえば BSP では 1 ビット，QPSK では 2 ビットが 1 シンボルとなる．シンボルレートで変調後の信号帯域が決まる．

白色ノイズ

ランダムノイズで，スペクトルで見ると平坦な特性をしている．そのため白色ノイズと呼ばれている．システムの性能評価などに使われる．ハード的には，熱雑音などを利用した発生器がつかわれる．ディジタル的には疑似ランダム系列（PN 系列）が使われる．

ドップラー効果

高速移動体から発信された電磁波や音波が，その速度に比例して周波数偏移を起こす現象．たとえば衛星通信では，その周波数偏移を補うような，PLL 処理などを必要とする．また，これを逆に利用して，移動体の移動速度を測ることができる．

フェージング

受信信号が，時間とともにその振幅や位相が変動する現象．無線機の設計の際に，フェージング耐力が重要な設計パラメータの一つとなる．周波数に関係なく一様にレベル変動する場合と，周波数によって変動のパターンが異なる周波数依存性フェージングがある．

プリアンプル

データ通信では，データを一つの集まり（パケット）として送る．その先頭部分に，データとは直接関係ない，同期のためのデータが付加される．この部分を「プリアンプル」と呼ぶ．回線等価のためや，キャリア再生などのために使われる．

マルチパス

送信機からの電波が複数経路を通り受信機にたどり着いたとき，それらがお互いに干渉を越す現象．高速通信になればなるほど受ける影響は大きく，移動体無線などで，深刻なデータエラーを引き起こすことがある．テレビでは，ゴーストの原因となる．

離散フーリエ変換

フーリエ変換は，$-\infty$ から ∞ までの畳み込み積分から計算される．これを計算機で計算する場合は，データはサンプリングされた有限個の値となる．そこで，ある区間の信号が無限に繰り返すと仮定して有限区間のサンプルのみのデータの畳み込み積和を，サンプリングされた信号のフーリエ変換とするもの．

60GHz帯を使った高速無線伝送技術

GaAs

ガリウムヒ素．Ga(ガリウム)とAs(ヒ素)から成る化合物半導体で，おもにトランジスタの半導体材料などに用いられる．電子の移動度がSiと比較して数倍も速い．GaAsトランジスタはおもに衛星放送受信用，携帯電話などのアナログ高周波通信用用途として普及している．

ISMバンド
(Industrial, Scientific and Medical Band．産業科学医療用バンド)

電子レンジや医療用加熱装置など，電波のエネルギを直接利用する特別な装置のために割り当てられた無線周波数帯．通信に使う場合は無線免許なしで自由に使えるが，通信品質の保証はない．

MMIC
(Monolithic Microwave Integrated Circuit)

高周波(マイクロ波)帯で動作するガリウムヒ素，あるいはシリコン集積回路(IC)を指す．携帯電話などで，電波の送受信に多く使われている．

PLL回路
(Phase Lock Loop回路)

負帰還回路によって周波数および位相を基準信号もしくは所望の信号と同一に安定化させた信号を得るための回路の一つ．おもに同期検波のために使用され，発振器，位相比較器，乗算器，ループフィルタなどから構成される．

UWB(Ultra Wide Band)伝送方式

無線通信の方式の一つで，データを比帯域20%以上のきわめて広い周波数帯に拡散して送受信を行うもの．それぞれの周波数帯に送信されるデータはノイズ程度の強さしかないため，同じ周波数帯を使う無線機器と混信することがなく，消費電力も少ない．UWBは位置測定，レーダー，無線通信の三つの機能を併わせ持っており，きわめて独特な無線応用技術といえる．

100Base-TX

IEEE802.3が規定する非シールドより対線(UTP)を使う100Mbps Ethernetの物理層仕様の一つ．名称の最後の"X"は，使用するより対線の仕様がANSI X3T9.5分科会が規定したFDDI/CDDIを基にしていることを示す．使用するケーブルは，カテゴリ5の2対(4心)UTPである．

アンテナ空間ダイバーシティ受信

二つ以上のアンテナを空間的に離して配置し，各々からの受信信号を切り替える，もしくは合成することで高品質受信を実現する方法．おもに移動受信におけるフェージング対策に用いられる．

加入者系無線アクセスシステム

加入者系無線アクセスシステムは，「WLL(Wireless Local Loop)」や「LMDS(Local Multipoint Distribution Service)」など，さまざまな名称で呼ばれてきたが，ITUの1997年世界無線通信会議で「Fixed Wireless Access(FWA)」を統一用語として採用することになった．

コスタスループ回路

PLL回路と同様に，負帰還回路によって周波数および位相を基準信号もしくは所望の信号と同一に安定化させた発振信号を得るための回路．おもに同期検波のために使用され，発振器，位相比較器，乗算器，ループフィルタなどで構成される．

周波数ドリフト

発振器やRFの周波数が漂動すること．

スーパーヘテロダイン伝送方式

マイクロ波帯などの高周波を使用する無線通信方式において，送信回路は中間周波数帯で送信変調信号を生成した後，これを局部発振信号を用いて無線周波数帯の信号へ周波数変換してアンテナより送信し，逆に受信回路は受信した無線周波数帯の変調信号を局部発振信号を使用して一度中間周波数帯へ周波数変換した後，復調することを特徴とする無線伝送方式．

多相位相変調方式

M-array PSK(Phase Shift Keying)変調方式．限られた周波数帯域で比較的多くの情報を伝送することのできる変調方式の一つ．多値数としては4相，8相のものが主として使用されている．とくに4相PSKのことを「QPSK(Quadrature Phase Shift Keying)変調」と呼ぶ．

多値直交振幅変調変調方式

M-array QAM(Quadrature Amplitude Modulation)変調方式．限られた周波数帯域でもっとも多くの情報を伝送することのできる変調方式の一つ．多値数としては16値，64値などがおもに使用されており，最近では256QAMや1024QAMなども開発されている．しかし，増幅器などによる非線形歪みに弱いことから64値以上はあまり用いられていない．

マイクロ波帯

およそ1GHz(波長30cm)～30GHz(波長10mm)の電磁波．UHF波との境界域(1～3GHz)を「準マイクロ波」と呼ぶこともある．

マルチパス

一般に電波が複数の経路をたどって受信点に到達することをいう．これによって直接届く波(直接波)と遅延して届く波(遅延波)が干渉してデータ誤りの原因となる．地上ディジタル放送波に使用される予定のOFDM変調方式はこのマルチパス妨害に強い方式として知られている．

ミリ波自己ヘテロダイン伝送方式

送信回路が局部発振信号成分を変調信号とあわせて送信し，受信回路はこれを自乗検波することで復調する，通信総合研究所が開発したミリ波帯無線伝送方式．ミリ波帯で実現が困難な高周波数安定な発振器を必要としないことと，受信回路に発振器やキャリア再生回路を必要としないことから，ミリ波帯システムの低コスト化と高品質伝送を実現する無線伝送方式として期待されている．

ミリ波帯

30GHz(波長1cm)～300GHz(波長1mm)の電磁波．波長がmm単位となることから，このように呼ばれる．マイクロ波帯との境界域(およそ20～30GHz)を「準ミリ波」と呼ぶこともある．

誘電体共振型発振器

Dielectric Resonator Oscillators(DRO)．「誘電体発振器」とも呼ばれる．誘電体共振器(DR)を使用して実現する発振器．

超広帯域(UWB)ワイヤレス通信の基礎と動向

BPSK(Binary Phase Shift Keying)

ディジタル変調方式の一種．無線通信の際

に電磁波の振幅，位相，周波数などに変化をもたせることにより，情報をのせることができる．BPSKとは，位相方向に情報を1シンボルあたり2ビットのせることが可能な位相変調方式である．実際にはQPSK（1シンボルあたり1ビット）や16QAM（1シンボルあたり4ビット）など，より情報速度を高めることの可能な変調方式が実用化されている．すなわち，BPSK変調方式では安定した通信は望めるが，高速伝送は基本的にめざしていない．

DS-SS方式

CDMAに用いられる手法の一つ．信号を元の情報信号帯域幅より十分速いチップレートのディジタル符号系列（拡散系列）で変調する方法．このとき，拡散系列がユーザーの識別に用いられる．各系列は互いに相互相関が低い（ユーザーの識別のしやすさ）こと，種類が多い（ユーザー数）ことなどが求められる．DS-SSによる通信のようすを**図3**に示す．ここではユーザーAとユーザーXの通信を目的とし，干渉としてユーザーBがいる状況を示している．2次変調である拡散変調において，ユーザーA，Bは異なる拡散系列を用いている．これより，ユーザーXが拡散系列に対する復調（逆拡散）を行った場合，ユーザーAの信号は見えるがBの信号は雑音と同様の特性を見せることで通信が確立できることを示している．

FCC（Federal Communications Commission）

「米連邦通信委員会」と呼ばれる合議制の独立規制委員会で，ラジオ，テレビ，電話，ケーブルテレビ，衛星放送を含む電気通信事業を規制監督する連邦政府機関のこと．近年，無線通信の形態がUWBなどによって大幅に変わってきたため，従来の無線通信に対するルールを適用することが困難になってきている．それらに対して対策を打ち出しているが，完全なる解決には至っていない．

FH

符号分割多元接続の一種．各ユーザーに異なる系列を割り当て，その系列によって周波数方向へ用いる帯域を切り替える．これが「周波数ホッピング」と呼ばれるゆえんである．その利点は，競合する符号分割多元接続の一種であるDS-SSのもつ遠近問題が存在しないことである．

IEEE802.11a/b/g

無線LANの諸規格．当初，802.11b（11Mbps，2.4G）が普及版として，それに対して高速化された802.11a（最大54Mbps，5.2G）が用意されていた．そこにbと互換性があり，かつ，aと同等の伝送速度で到達距離が延びるというふれこみで802.11g（最大54Mbps，2.4G）が出てきた．しかし，aとgの比較は非常に困難であり，実装の手法などによってその比較結果はまちまちである．また，bとの互換性に関しても，gとbが同時に使われる状況では，gは結果的にbと同様の伝送速度に落とす必要がある．

IEEE802.15.3a

パーソナルエリアにおける通信（Wireless Personal Area Network：WPAN）の標準化組織としてIEEE802.15委員会は結成された．この中で，とくに物理レイヤの通信方式の名称をIEEE802.15.3aという．現在も，その標準化作業は進行中であり，標準化案の中のじつに95％がUWB方式となっている．

IMT2000

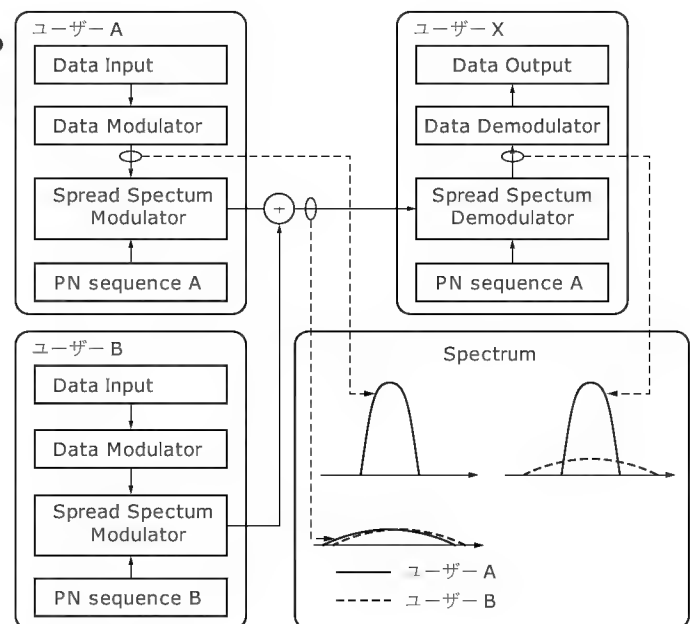
第3世代携帯電話規格のこと．日本ではNTT DoCoMoのFOMA，KDDIのCDMA 2000x1などがあげられる．これまでの規格は，日本のPDC，ヨーロッパ，アジア各国ではGSM，アメリカではDigital AMPS，IS-95がおもなものとしてある．このように国によって異なる規格を統一しようという動きの中で，IMT2000は生まれてきた．第2世代に比べて大幅に伝送レートが向上したことで，画像の受け渡し，携帯電話用アプリケーションの利用における性能の向上，より高精細な動画配信などが可能となった．

Ultra Wide Band（UWB）

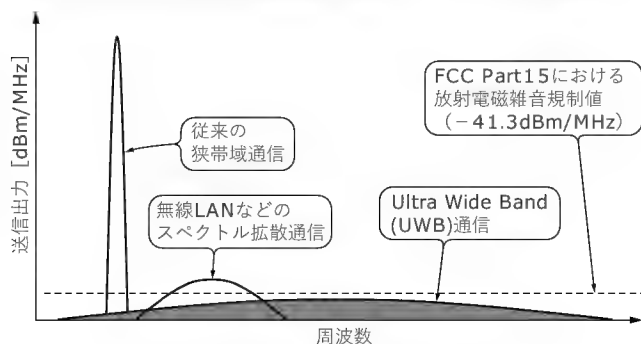
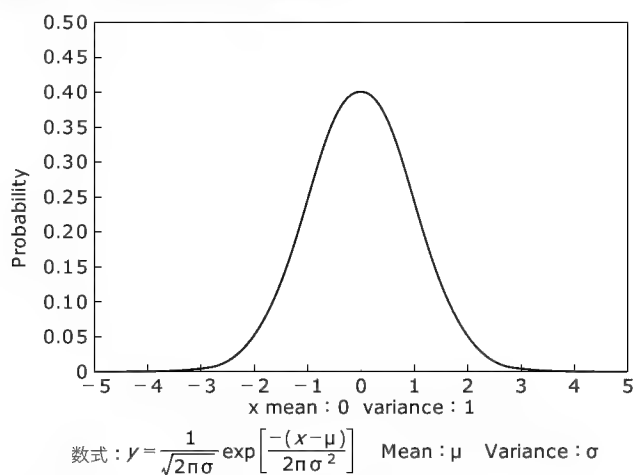
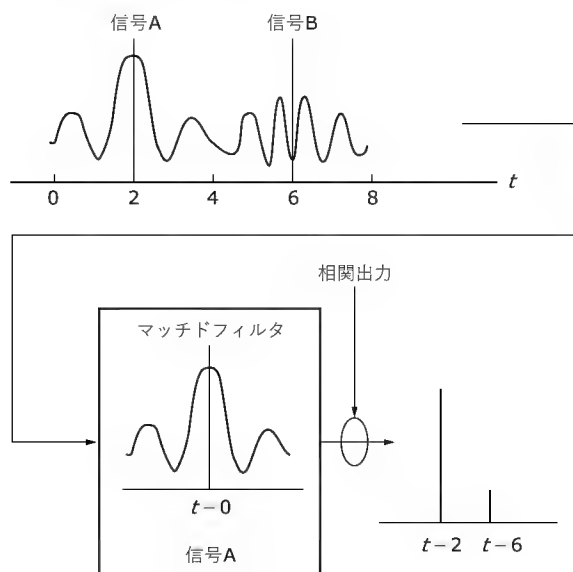
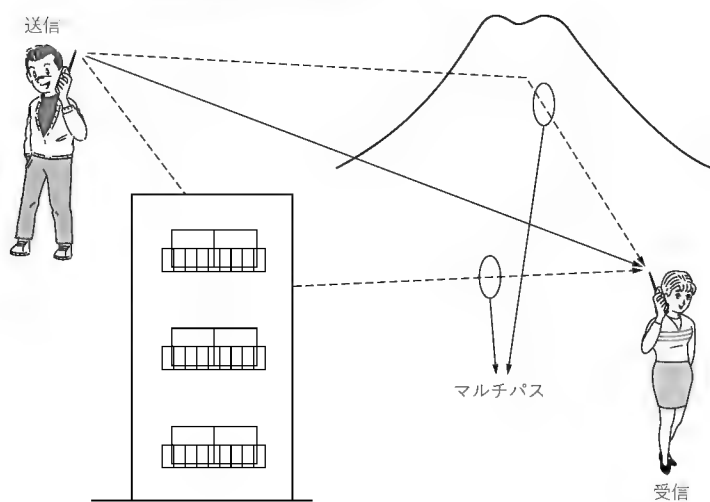
現在，もっとも注目を浴びている無線通信技術の一つ．「超広帯域無線」を意味し，中心周波数の25%以上，または1.5GHz以上の帯域幅を占有する無線伝送方式を指す．その際立つメリットは，超高速伝送（数百Mbps）を実現できることにある．その理由は従来，無線通信において用いられていた搬送波を用いずに1ns以下という短パルスを用いているところにある．結果的に，パルス幅の短さが高速伝送を生むしくみになっている．一方で，他の通信システムと干渉することが前提となっているため，その対策が大きな課題の一つとして挙げられる．**図4**に，UWB信号と従来の変調信号の電力スペクトル分布の相違を示す．

ガウス分布（正規分布）白色ガウス雑音

あるランダムなパラメータの確率的分布の一つとして，ガウス分布がある．その定義は，数式とグラフによって求まる．**図5**にそれらを示す．物理的意味としては「ガウス分布は偶然誤差の分布法則」といえる．すなわち，まったく予測されない出来事によって起きた偶然的差であるということである．また，無線通信においてそのようなランダムな雑音

〔図3〕DS-SSによる通信の概略図

〔図4〕UWB信号と従来の変調信号の電力スペクトル分布の相違

〔図5〕ガウス分布のグラフと数式

数式：$y=\frac{1}{\sqrt{2\pi\sigma}}\exp\left[\frac{-(x-\mu)}{2\pi\sigma^2}\right]$　Mean：μ　Variance：σ

〔図6〕マッチドフィルタの動作原理

〔図7〕屋外での無線通信におけるマルチパスの概略

を白色ガウス雑音として取り扱っている．そのスペクトルは，周波数軸上に無限に一様である．

スペクトル拡散信号

スペクトル拡散を文字のとおりに解釈すると，送信時に要求される周波数帯域幅より，ずっと広い周波数帯域幅に拡散することを指す．スペクトル拡散信号はすなわち拡散された信号を指し，その特徴を列挙すると，次のようになる．

1) 干渉(狭帯域通信からの混信やマルチパスフェージング，故意の妨害など)に対する耐性が強い
2) 電力スペクトル密度が低いので，狭帯域通信へ与える妨害の程度が低い
3) 雑音より電力スペクトル密度が低い状況でも通信でき，通信の秘匿性がある
4) 傍受からの回避が可能である
5) 異なる拡散符号を用いることにより複数の利用者が同一通信路にランダムアクセス(非同期CDMA)できる
6) 測位測距の能力がある

多元接続　タイムホッピング

複数の人間が同時に通信をする場合に，各ユーザーの切り分けを行う際の手法を指す．よく用いられる手法としては時間分割多元接続がある．これは各ユーザーが使う時間を切り分けて通信をすることで干渉を起こさせない方法である．ほかに周波数，符号，空間などがある．UWBにおいてはとくに時間方向にランダムにパルスを配置する系列を用意し，各ユーザーに異なる系列すなわち，タイムホッピング系列を割り当て，符号分割多元接続を実現する手法が検討されている．

パルスレーダー

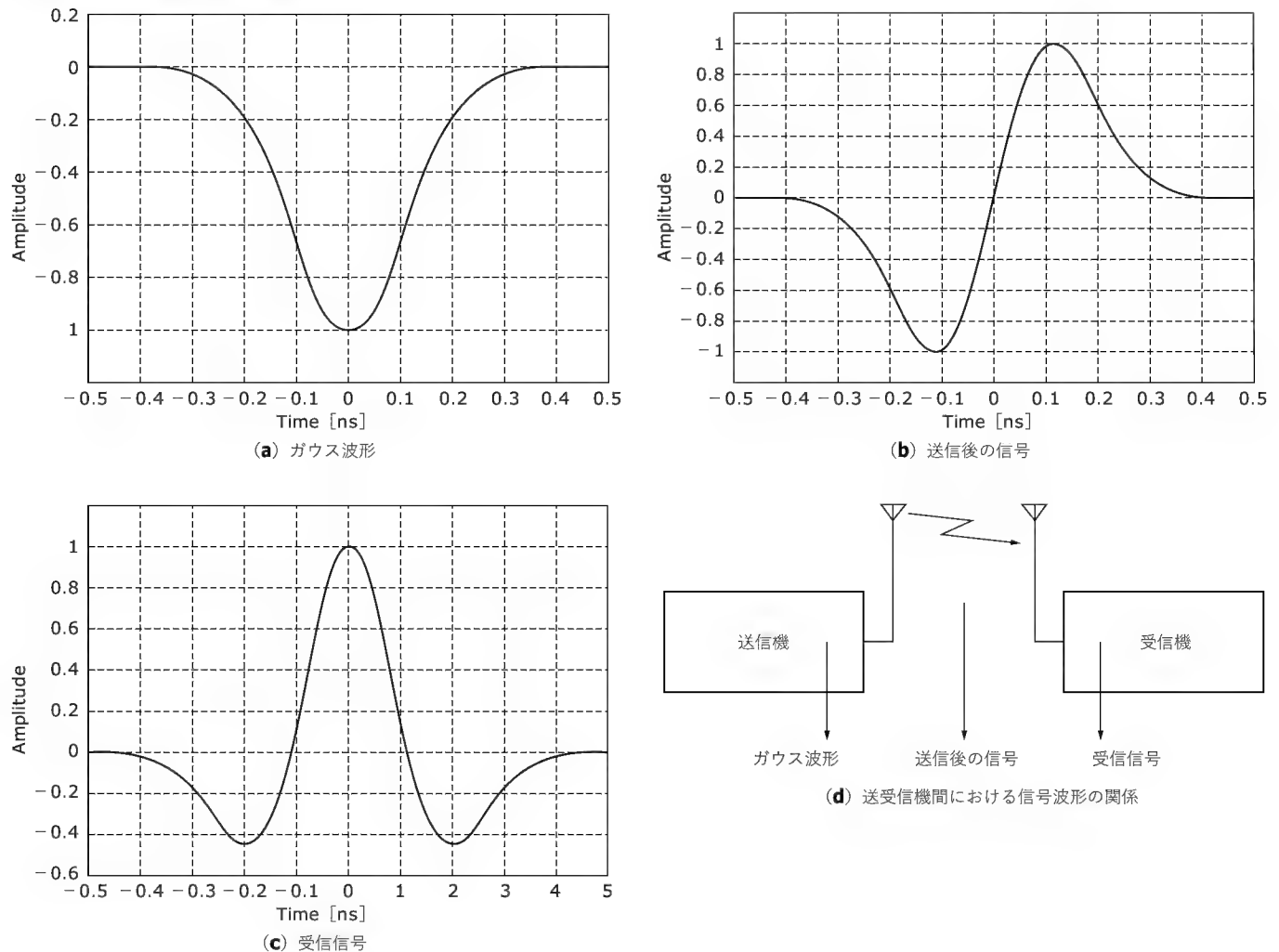
パルスレーダーとはレーダーの一種で，アンテナから送信されたパルスが目標物に反射され戻ってくるまでの時間を計測することにより，目標までの距離を測ることを可能とする電波探知装置である．近年注目を集めているUWB(Ultra Wide Band)レーダーもこのパルスレーダーの一種と考えることができ，ITS(Intelligent Transport System)などにおいて車車間，路車間の測距に検討されている．

〔図8〕送受信機間における信号波形の関係

(a) ガウス波形

(b) 送信後の信号

(c) 受信信号

(d) 送受信機間における信号波形の関係

● マッチドフィルタ　相関出力

このフィルタの役割は，簡単にいうと信号の類似性を調べるものである．受信機において受信機の求める形をフィルタによって表現し，受信機で受信信号とフィルタとの相関出力を見る．求める形と受信信号の形がまったく一致した場合に最大の値を出力することとなる．すなわち，相関出力は信号の類似度を示す指標となるわけである．これより，受信機では自分の希望信号であるかどうかを確認することが可能である．

図6にその動作原理を示す．ここでは受信機内に信号Aを取り出すことを目的としたマッチドフィルタがあるとする．よって，フィルタは信号Aをモデル化したものが用意される．そこで，信号Aが来た場合は最大の相関出力を得ることができ，Bが来た場合は低い相関出力となる．これより，受信機はAを受信できたことを確認できる．

● マルチパス

無線通信における最重要課題の一つ．**図7**にその概略図を示す．送信から受信まで空間を電磁波が通るとき，大きく二つに分かれ，障害物に反射して届くマルチパスと，そうでない直接波がある．無線通信の信号はこのマルチパスにより，信号の振幅位相変動，そして符号間干渉によって劣化させられることが大きな問題となる．同時に，この問題に対して無線通信のエンジニアは常に取り組んでいる．すなわち，無線通信の通信路をいかに有線と同等にするかを日々考えているわけである．

● モノサイクル

UWB-IRにおいて用いられる波形．とくに，UWBではガウス波形が用いられることが検討されている．このモノサイクル波形の特徴として時間パルス幅が小さいことが挙げられる．結果的に広帯域信号となり，メリットとして「高速通信」，「低消費電力」，「マルチパスに対する耐性」が生まれる．

図8に信号波形と送受信機間の関係を示す．信号がアンテナを通ることは波形を微分することと同値であり，その結果，図のように送信前のガウス波形，送信後の信号，受信信号というように波形が変化する．

阪田　徹　NTTアクセスサービスシステム研究所
中野敬仁　(株)アットマークテクノ
西村芳一　(株)エーオーアール
辻　宏之　独立行政法人 通信総合研究所
荘司洋三　独立行政法人 通信総合研究所
河野隆二　横浜国立大学大学院
梅林健太　横浜国立大学大学院

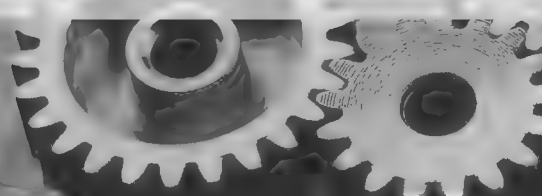
chapter 09

カードにCPUとOSが載った！

ICカード技術の基礎と応用

テレフォンカードや定期券などの基盤として，いままで使用されてきた磁気カードに代わり，ICカードが注目を集めている．単なる記憶装置にすぎなかった磁気カードと比べ，ICカードはCPUとメモリを搭載し，その中でOSを動作させることにより高いセキュリティを実現している．現在，ICカード用OSとしてさまざまなものが使われている．また，非接触カードと接触カードでは使用されるハードウェアが異なっている．

本誌2003年3月号の本誌特集「ICカード技術の基礎と応用」では，ICカードアプリケーションを開発するにあたって必要とされる基本知識から，開発の実際，応用例や今後の展望までを詳しく解説している．

（編集部）

ICカードOS「MULTOS」によるカードアプリケーションの作成

ICカード

CPU，メモリが搭載されたICチップが埋め込まれたカードをいう．カード表面にICチップが露出している「接触型」と，電磁波で通信するためICチップが露出していない「非接触型」に大別できる．磁気カードに比べて大容量，高セキュリティであることからクレジットカードや住民カード，交通系カードなどに広く使用されはじめている．

ISO/IEC14443

近接型と呼ばれている伝送距離が10～20cm程度の非接触方式ICカードの国際標準規格で，物理的特性や伝送プロトコルなどを定めている．通信方式によりヨーロッパやアジアで使用されているType A，日本の公共カードを中心に普及しているType Bがある．

ISO/IEC7816

CPU付き接触型ICカードの国際標準規格で，物理特性，端子位置，電送信号や伝送プロトコル，APDU（Application Protocol Data Unit）と呼ばれるコマンド形式や共通コマンドなどを規定している．

MAL（MULTOS Application Language）

MULTOSアプリケーションを開発するときに使用するアセンブリ言語．開発ツールを使用することにより，MULTOS実行言語であるMELに変換される．MULTOSのアプリケーション開発はこのMAL以外にC言語，Java言語での開発も可能である．

MAOSOコンソーシアム

MULTOSの技術仕様を運営管理している非営利団体で，MULTOS標準化仕様の策定機関である．参加団体名や活動詳細，各種オープンドキュメントの入手についてはMULTOS公式ページ（http://www.multos.com/）を参照のこと．

MEL（MULTOS Executable Language）

MULTOSアプリケーションのMULTOSの実行可能言語．ICカードでの使用のために最適化された構造化言語であり，限られたリソースを最大限に利用できる．

MULTOS（MULTi-application Operating System for smart cards）

マルチアプリケーションに対応したICカード用オペレーティングシステム（**図1**）．オープンな仕様であり，かつ高セキュリティであることから，金融系ICカードや公共ICカードなどで幅広く普及している．

MULTOS鍵管理局（MULTOS Key Management Authority）

MULTOSカードの発行に必要なカード公開鍵証明書やアプリケーションをロードするためのロード証明書などを発行する認証局．英国に，日本を含む全世界に向けサービスを提供するGlobal KMAがあり，日本国内専用には（株）日本スマートカードソリューションズ（http://www.jssco.net/）がある．

楕円曲線暗号（ECC：Elliptic Curve Cryptography）

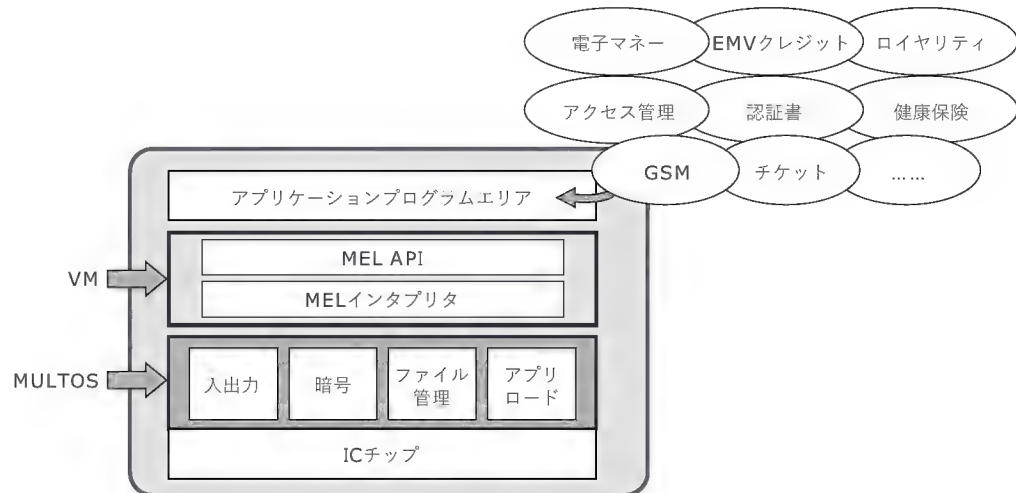
RSA暗号方式に代わる公開鍵暗号方式であり，RSA暗号より短い暗号鍵長で同等の暗号強度を実現している．このため，RSAに比べ処理速度の向上，メモリの削減，消費電力の減少が期待でき，モバイル機器やICカードに向いている．

非接触方式

カードからデータを読み取るために金属端子をICチップに接触させる必要のある接触型方式に対し，リーダ/ライタ端末にカードをかざすだけでデータを読み取ることが可能な方式．交通系カードや入退室カードなどに用いられている．

マルトス推進協議会

MULTOSの普及とその利用環境を向上させるために日本国内向けに設立された非営利団体で，国内でセミナー・研修などの活動を行っている．活動内容の詳細については，マルトス推進協議会公式ページ（http://www.multos.gr.jp/）を参照のこと．

〔図1〕MULTOSの特徴

ICカードOS「ASEPcos」での開発とセキュリティ

APDU (Application Protocol Data Unit)

ICカードと端末間で送受信されるデータのフォーマット．ICカードに対して送られるAPDUをコマンドAPDU，ICカードから返されるAPDUをレスポンスAPDUと呼ぶ．コマンドAPDUはクラスバイト，インストラクションバイトおよびパラメータからなるヘッダ部と，データ長フィールドとデータフィールドからなるボディ部で構成される．レスポンスAPDUは，データフィールドと処理結果を示すステータスバイトで構成される．

CSP

個々のICカードの違いを吸収し，アプリケーションプログラムに対して統一されたインターフェースを提供する機能をサービスプロバイダと呼ぶ．ICカードの一般的な機能については，ICCサービスプロバイダが，暗号に関する機能についてはクリプトサービスプロバイダ(CSP)が用意される．

ISO14443標準

非接触型ICカードに関する国際標準．符号化の方式によりType AとType Bが定められている．

ISO7816標準

ICカードに関する国際標準．現在パート1からパート9が標準化されている．パート1からパート3では，接触型ICカードの物理的・電気的特性および通信プロトコルが定義されている．パート4以降は，コマンドやセキュリティなどが定義される．パート3で定義される通信プロトコルのうち，キャラクタ転送を行うものをT＝0，ブロック転送を行うものをT＝1と呼ぶ．

JICSAP仕様2.0版

有限責任中間法人日本ICカードシステム利用促進協議会が制定する，ICカードの実装規約．ISOに準拠しており，国内におけるデファクト標準として認知されている．2.0版では，従来からの接触型に加え，非接触型についても標準化を行っている．

PIN (Personal Identification Number) コード

カードの所有者を照合するためのいわゆる暗証番号．ICカードでは，キーのうち暗号技術を利用せずに平文で照合されるキーをPINと呼ぶこともある．

RSA公開鍵暗号アルゴリズム

現在もっとも広く利用されている公開鍵暗号アルゴリズム．Rivest，Shamir，Adlemanという発明者3名のイニシャルをとってRSAと命名された．二つの「大きな」素数のかけ算はたやすいが，その結果から元になった素数を導き出すのはきわめて難しいという原理によっている．

SHA-1ハッシング

一般に，元のデータ長に関わらず，常に固定長のデータを生成するアルゴリズムをハッシュと呼ぶ．ハッシュでは，元のデータが1ビットでも変更されると，まったく異なった結果が得られる．この特性を活かし，電子署名で署名対象となるメッセージダイジェストの生成に使用されている．SHA-1は，1992年にNISTが発表したSHAの改良版で，1995年に発表されている．

公開鍵暗号アルゴリズム

暗号化と復号化にそれぞれ異なる一対のキーを使用する暗号アルゴリズム．一対のキーのうち一方を厳重に管理しておけば，他方は公開してもセキュリティにまったく影響を与えないため，公開鍵と呼ばれる．セッションキーの配布や，電子署名などに使用される．

接触型

ICカード表面に搭載されたコネクタを通して電源，クロックの供給および入出力が行われる．コネクタの位置および電気的な定義についてはISO7816-2で標準化されている．古くから実用化されている技術で，現在金融分野や道路公団のETCなどで採用されている．

非接触型

接触型と異なり，コネクタをもたず無線を使って通信するICカード．電源は，電磁誘導で供給される．リーダに触れる，あるいはかざすだけで動作するため，一定時間に大量のカードを処理することが可能で，JR東日本の「Suica」などの交通機関や建物への入退出管理などで利用が進んでいる．

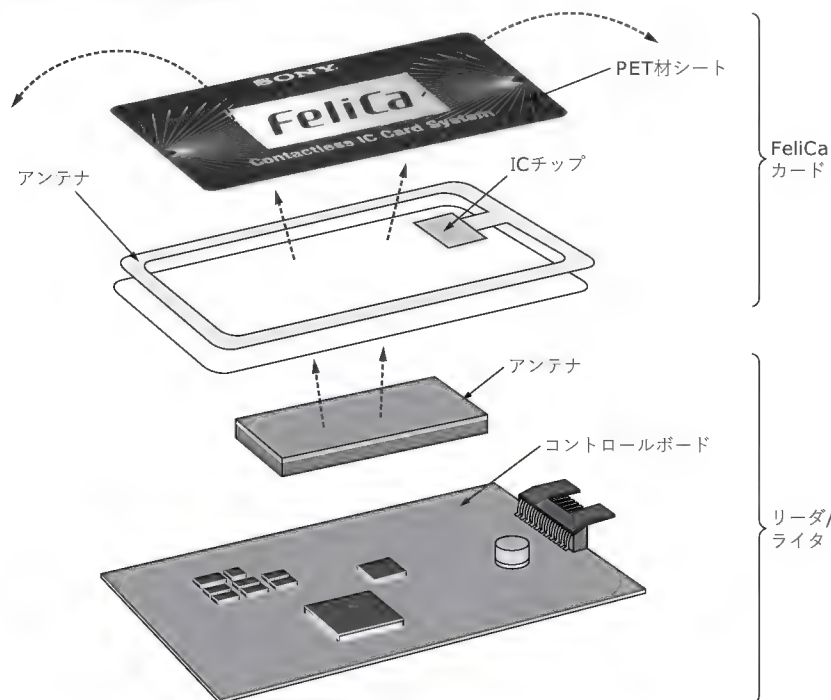
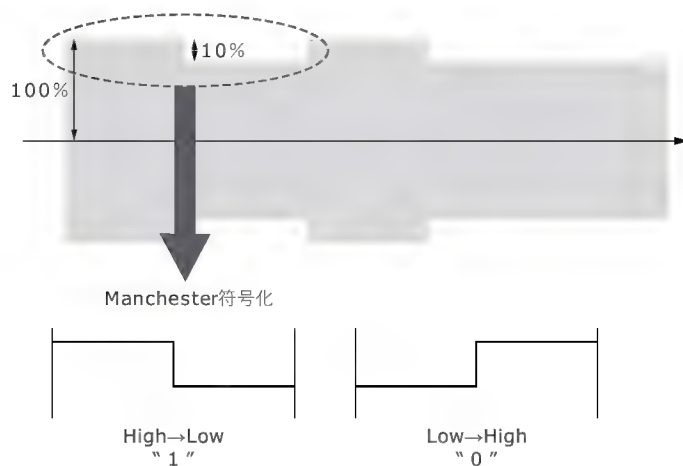
メモリスペースのフラグメンテーション解消

EEPROMなどの不揮発性の記憶媒体にデータの書き込み，消去を行っていると，しだいに媒体の空き領域が虫食い状態のように散在するようになる(フラグメンテーション)．この状態では，総空き容量(空き領域の合計)に比べ実際に使用できる空き容量が少なくなり，メモリの使用効率が低下するという問題が発生する．ICカードのように記憶容量に制限のある媒体では，このフラグメンテーションを解消することがとくに重要とされる．

非接触ICカード技術「FeliCa」の概要

FeliCa(フェリカ)

ソニーが開発した非接触ICカード技術方式．「Felicity：至福」から発展させた名前のとおり，日常生活をより楽しく便利にするために誕生した．現在ではとくに，各国の交通機関でのICカード乗車券として広く利用されている．FeliCaの内部構造を**図2**に示す．

〔図2〕FeliCaの内部構造

〔図3〕Manchester方式

FeliCaOS

FeliCa技術のうち，とくにコマンド処理およびファイルシステムをつかさどる部分をFeliCaOSと呼ぶ．ファイルごとのセキュリティ設定機能，複数ファイルの同時アクセス機能などの特徴をもち，かつ非接触ICカードとしてトランザクションの高速処理が可能．

FeliCa相互認証方式

FeliCa技術方式で利用する独自相互認証方式で，アクセスする複数のサービス(ファイル)の鍵から相互認証用の鍵を生成し，この鍵を利用して認証を行う方式のこと．これにより，セキュリティを確保しつつ高速な認証処理が可能となっている．

FeliCa無線通信インターフェース

FeliCa技術の一部として規定される，カードとリーダ/ライタとを直接接触することなしに無線でデータの送受信を行う通信方式を指す．無線通信は，13.56MHzの周波数帯を利用し，212kbps以上の速度で行われる．副搬送波を使用しない「対称通信」が特徴．

ISO/IEC14443

ICカードは，国際標準化機構(ISO)，国際電気標準会議(IEC)においてJTC1/SC17という委員会で国際的な標準化作業が進められている．非接触ICカードの国際標準規格はISO/IEC14443として規定されており，内容によりPart1からPart4まで規定されている．

Manchester符号化方式

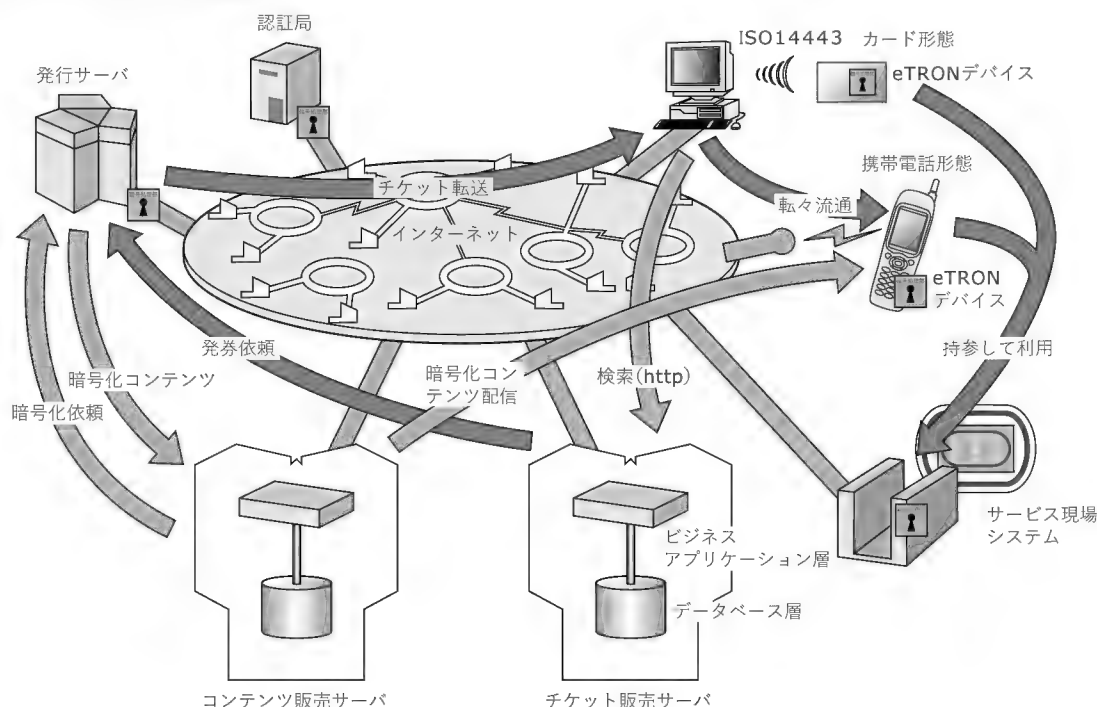
データを0と1とで表現するように符号化する方法の一つ．たとえば，ビット区間の中央で電圧レベルを「高」から「低」へ変化させることで「1」を表現し，逆に電圧レベルを「低」から「高」へ変化させることで「0」を表現する(**図3**)．

MIFARE

Philips社が主体となって推進している非接触ICカード技術の一つ．

NRZ符号化方式

データの0と1を単純に信号波形の高低にそのまま対応させる，もっとも一般的な方式で，たとえば0を「低」，1を「高」として電圧などの高低でディジタルデータを表現することを可能とする符号化方式．

〔図4〕eTRONアーキテクチャの全体像

PET材〔PET:ポリエチレンテレフタレート(poly ethylene terephthalate)〕

燃しても有害なガスが発生しない，環境ホルモンの疑いがない，軽くて割れにくい，加熱して融かせばリサイクルしやすい，などの特徴から，ビニール材の代替素材として使用されてきている．飲料容器(PETボトル)の原料としても利用されている．

Triple-DES暗号

DES暗号アルゴリズムを3重に適用するようにした方式のこと．コンピュータの性能向上にともなってDES暗号を解読される危険性が高まったため，同じ方式を3重にかけることにより，強度を高めている．二つの鍵を利用する方式と，三つの鍵を利用する方式の2種類がある．

Type A方式

ISO/IEC14443で規定されている近接型非接触ICカードの方式の一つ．変調方式はASK100％，エンコード方式はModified MillerおよびManchester，基本的な通信速度は106kbps，などの仕様をもつ．

Type B方式

ISO/IEC14443で規定されている近接型非接触ICカードの方式の一つ．変調方式はASK10％，エンコード方式はNRZ，基本的な通信速度は106kbps，などの仕様をもつ．

3パス相互認証方式

2者間で通信を行う際に，まずはじめに相手が正しいかどうかを相互に確認する処理を相互認証と呼ぶ．とくに3パス相互認証方式は，2者間において，3回のやりとりで相互認証を完結させる．

近接型

非接触式ICカードは，データの読み書きのできる距離によって，次の三つのタイプに分けられる．

- 密着型(～2mm，通信速度9.6kbps～，周波数4.91MHz)
- 近接型(～10cm，通信速度106kbps～，周波数13.56MHz)
- 近傍型(～70cm，通信速度～26kbps，周波数13.56MHz)

近接型非接触ICカード(Proximity IC Card)は，13.56MHzの周波数を利用する方式として，ISO/IEC14443として標準化されている．

クローズドエリア

ICカードなどの運用開始にあたって，不特定多数にサービスを提供するのではなく，テーマパークや大規模複合施設など，ある閉ざされた特定エリア(Closed Area)での運用を想定する場合，クローズドエリアでの運用と呼ぶ．

eTRONの概要

eTRON(Entity and Economy TRON)

TRONはThe Realtime Operating system Nucleusの略．電子媒体で表現された価値情報や電子実体(Entity)をセキュアに扱う総合的なシステム体系(図4)．

eTRON/16

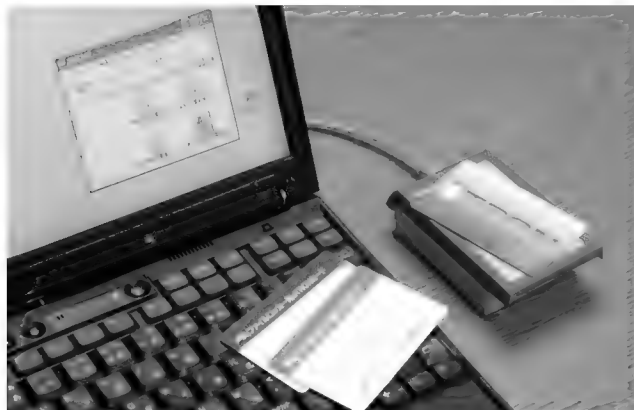
16ビットマイコンクラスのCPUをもった小型チップによって実現できるeTRONの規格．接触と非接触通信の両方をもつデュアル通信機能や，公開鍵暗号系アルゴリズムによる暗号認証通信機能をもち，SIMチップ形状に作られている．

eTRON/32

32ビットマイコンクラスのCPUをもった小型チップによって実現できるeTRONの規格．接触と非接触通信の両方をもつデュアル通信機能や，公開鍵暗号系のアルゴリズムによる暗号認証通信機能をもち，SIMチップ形状に作られている．

eTRON/8

8ビットマイコンクラスのCPUによって実

〔写真1〕
ファイルロッカー

現できるeTRONの規格．非接触通信機能や暗号認証通信機能をもち，カード形状に作られている．

eTRON/8 SDK

eTRON/8 SDK（Standard Developers Kit）は，eTRON/8を利用した，セキュアアプリケーションを開発するための標準開発キットである．パーソナルメディア社から発売されている．

eTRON/T（eTRON/Terminal）

ディスプレイや入力デバイスを備え端末形状をしたeTRON機能を有するコンピュータのためのeTRONの規格．

eTRON ID

eTRONで使われる端末やチップをユニークに識別するための識別子（identifier）の規格．製造時に付与された128ビット長データで，ユーザーが変更することはできない．

ISO 14443

カードの内部にアンテナをもち，外部の端末が発信する弱い電波を利用してデータを送受信する，非接触型ICカード（近接式）の国際標準規格がISO 14443である．

RFID（Radio Frequency Identification）

IDチップおよびアンテナから構成され，ICチップ内の情報を非接触で読み出し，書き換えが可能である．

T-Engine

組み込みリアルタイムシステムのための，ハードウェアおよびソフトウェアの標準アーキテクチャで，トロンプロジェクトが推進している．現在，130社あまりの企業や大学などとの共同研究開発プロジェクトとなっている．

電子マネーとしての利用（決済系）

決済処理の効率化などを目的として，通貨を電子的に扱うことが電子マネーとして注目されている．電子化情報は，品質を劣化させずに，複製，変更などが行えるため，安全性をどのように達成するかが，電子マネーの大きな課題である．eTRONはそうした電子マネーに用いることができる．

ファイルロッカー

PC上のファイルを暗号化して保存することによって，PC上の情報の盗難による情報漏えいを防ぐ，パーソナルメディア社の製品（**写真1**）．暗号のための鍵をeTRON/8カードに格納し，他の情報と別々に管理できるため，盗難に対して高い安全性を提供できる．

宇田川真理　（株）日立製作所 IDソリューション統括本部
進藤雄介　（株）日立製作所 IDソリューション統括本部
小坂　優　（株）アテナ・スマートカード・ソリューションズ
松尾隆史　ソニー（株）ネットワークアプリケーション＆コンテンツサービスセクター FeliCaビジネスセンター
坂村　健　東京大学教授/YRPユビキタス・ネットワーキング研究所所長
越塚　登　東京大学助教授/YRPユビキタス・ネットワーキング研究所副所長

chapter 10

480Mbps対応USBターゲットからホストシステムの設計まで

解説！ USB徹底活用技法

本章では，パソコンの世界で急速に普及の進む「USB2.0」と，パソコン以外の機器に実装されるようになってきた「USBホスト機能」という大きな二つの話題性をもつUSBに関する用語を解説する．USB2.0とUSB1.1の大きな違いは，そのデータ転送性能にある．USB2.0の採用により，10Mバイト/秒を超える高速転送を実現できる．また，たとえばいままでパソコンに対してターゲットとして動作してきたPDAが，デジカメやプリンタなどを接続するためのUSBホスト機能を実装しはじめている．この流れは，さまざまな組み込み機器に波及してきている．

本誌2003年4月号特集「解説！ USB徹底活用技法」では，10Mバイト/秒を超えるUSB2.0ターゲットシステムの設計事例や，組み込み向けホストコントローラ，USBプロトコルスタック，USBアナライザの活用法などをくわしく解説した．

（編集部）

USB2.0対応コントローラEZ-USB FX2の詳細/高速転送対応USBターゲットの設計事例

8051CPUコア

Intelが1980年に組み込み用途向けに開発した8ビットのマイクロコントローラ．多少癖のあるアーキテクチャではあるがコンパクトであり，Intelがライセンスしたこともあって，内蔵用CPUとして広く利用されている．

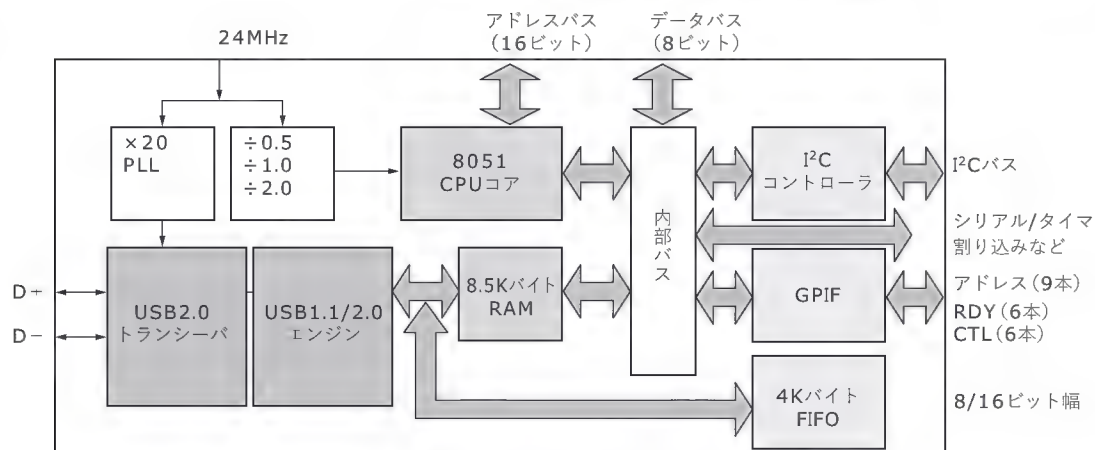
EZ-USB FX2

8051をコアにして，プログラムメモリやUSB2.0ターゲットコントローラを内蔵させたCypressのEZ-USBファミリ．プログラムは外部から内部RAMにダウンロードして動くほか，GPIFによる柔軟で高速な転送動作が可能．EZ-USB FX2の外観を**写真1**に，ブロック図を**図1**に示す．

FX2間伝送プログラム

FX2ボードを二つ対向接続してホストA→(USB2.0)→FX2→FX2→(USB2.0)→ホストBという接続で，ホストAからホスト

〔写真1〕
EZ-USB FX2(CY7C68013)
の外観

〔図1〕 **FX2のブロック図**

Bへの転送実験を実施．今回（本誌2003年4月号特集）はホストを複数用意できなかったため，単一ホストの別ポートを利用した．

GPIF

EZ-USB FXやFX2に搭載されたプログラマブルな波形パターン生成機構．内部のエンドポイントFIFO関係のフラグや外部からのステータス入力によって動作を決定でき，CPUで行うよりもはるかに高速な伝送を可能とした．FX2のGPIFのブロック図を**図2**に示す．

GPIFによるシングルリード/ライト動作の設計

GPIFによる転送のうち，CPUによって1アクセスごとにリード/ライトを行うモードをシングルリード/ライトという．GPIFはCPUからのシングル動作開始を指示されると，1回の転送動作を行って停止する．

GPIFによるバーストリード/ライト動作の設計

エンドポイントバッファ分の転送をまとめて行う動作モード．FX2のマニュアルでは「FIFOリード/ライト」と呼んでいる．混乱を避けるため，記事では表現を変えた．

IDEハードディスク

IDEは現在PCなどでもっとも一般的に使われているハードディスクインターフェースである．もともとはIBMのタスクインターフェースが基となって改良されたもの．その後大容量化や高速化に対応した拡張が行われ，現在に至る．

IFCLK端子

GPIFの動作クロック入出力端子．GPIFは，FX2内部で生成される30MHzか48MHzのクロック，または外部からのクロック入力に同期して動作する．外部クロック時のIFCLKは入力，内部クロック時のIFCLKは出力として利用可能．

I²Cコントローラ

Philipsが提唱した2線式のシリアルインターフェース．1本のI²Cバス上にRAM以外にも複数のさまざまなデバイスが接続可能．当初はクロック100kHzと400kHzのモードのみだったが，Version2.0で3.4MHzモードが定義された．

PCF8574

Philips社のI²Cバス用I/Oポートコントローラ．I²Cバス経由で8ビットの入出力ポートを制御できる．割り込み出力ピンもあり，I²Cバスによって汎用I/Oポートを簡単に増設できる．CypressのFX2評価ボードにも使用されている．

USB2.0

USB1.0/1.1の上位にあたる規格．1.0/1.1では1.5Mbpsのロースピードと12Mbpsのフルスピードの2種類が定義されていたが，2.0では下位互換を保ちながら480Mbpsのハイスピードを定義．より高速大容量の転送が可能．

USB-IDE/ATAPI変換アダプタ

IDE/ATAPIデバイスをUSBデバイスとしてみせかける変換アダプタ．USBのマスストレージクラスに対応させ，クラス定義のコマンドを受けてIDE/ATAPIコマンドに変換してデバイスにアクセスするものが一般的．

USBコントローラ

USBトランシーバと，ターゲットCPUの間を取り持ち，USBホスト（PCなど）から送られてきたデータやデバイスリクエストを受信してエンドポイントバッファへ格納したり，エンドポイントバッファの内容を送信する．

USBトランシーバ

USBバスと，USBコントローラの間でUSBバス信号とのレベル変換などを受け持つ部分．USB2.0のトランシーバはUSB1.1との互換性をもつ．外付けのトランシーバチップもあり，PHY（物理層）チップと呼ばれている．

USBハードディスク

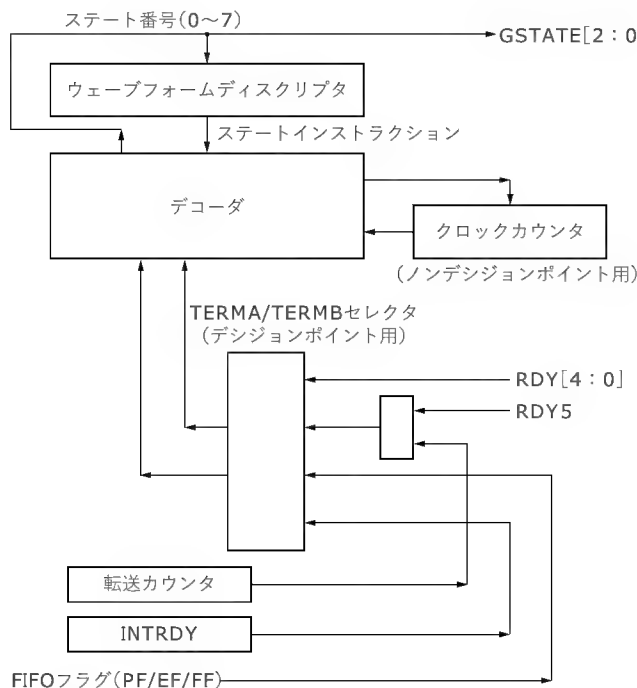
USB経由でコマンドやデータ転送を行うハードディスク．通常USBのマスストレージクラスの規格に準拠したUSB/IDE変換アダプタとIDEハードディスクで実現されている．低消費電力のHDDを使い，バスパワーで動作できるものもある．

ウェーブフォームディスクリプタ

FX2のGPIFの動作定義用のテーブル．GPIFのもつ四つの動作モードに対応して四つのディスクリプタをもてるようになっている．GPIFは起動されると該当するテーブルの内容を読み出し，データ入出力動作を実行する．

エンドポイントバッファ

USBのホストとUSBターゲットのCPUの間で，データをやりとりするためのバッファメモリ．バスアドレスとエンドポイント番号によって選択されたバッファメモリが

〔図2〕GPIFの構成概略

〔図3〕ウェーブフォームディスクリプタとステートインストラクションの関係

〔図4〕スレーブFIFOとGPIF

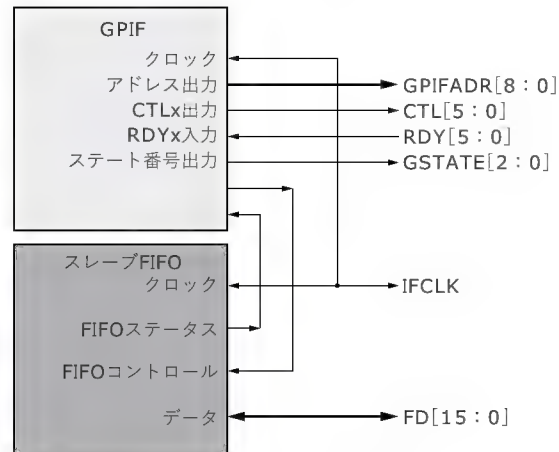
(a) スレーブFIFOモード時

(b) GPIF使用時

リード/ライトされることで，USB伝送が行われる．

ステートインストラクション

ウェーブフォームディスクリプタ中の，各ステートごとの動作定義の内容をCPUが実行する命令(インストラクション)になぞらえて，「ステートインストラクション」と呼ぶ．出力ピンの状態や，条件判定・分岐，ウェイト数などを定義できる．ウェーブフォームディスクリプタとステートインストラクションの関係を**図3**に示す．

スレーブFIFO

FX2のエンドポイントバッファは，チップ外部に対してあたかもFIFOメモリがついているように見せかけることができるようになっている．外部回路がマスタとしてリード/ライトを行うことから，スレーブFIFOと呼んでいる．スレーブFIFOまわりの接続関係について，GPIFを使わない場合と使った場合を**図4**に示す．

デシジョンポイント

外部入力やFIFOフラグの中から二つまでを選び，その論理演算結果によって次にどのステートに移動するのかを決定するステート．エラー処理や，あるフラグが立つまで現在のステートに留まらせるような場合に利用される．

トランザクションカウンタ

GPIFによる転送回数カウンタ．GPIFは内部のFIFOフラグなどのほか，CPUが設定したカウント数に達したときに停止させることができる．ステート7に戻るごとに転送カウンタはデクリメントされ，0になった時点で停止する．

内蔵RAM

組み込み用途ではスペースの制約も厳しいことから，RAMを内蔵した組み込み用CPUが多い．このRAMを「内蔵RAM」と呼ぶ．ソフトウェア的にはメモリ空間に置かれた通常のRAMと同じだが，アクセス速度は外部RAMより速いのが一般的．

ノンデシジョンポイント

現在のステートに一定時間留まってから次のステートに移行するステート．留まる時間はGPIFの動作クロックの1クロック単位で最大256クロックまで設定できる．パルス幅やセットアップ/ホールド時間を確保したいときなどに利用される．

バースト転送時の自動桁上がり

今回(本誌2003年4月号特集)の回路では，GPIFが出力する9ビットより上のアドレスはI/Oポートによる固定出力である．このため01FFhの次は0200hではなく0000hになる．0200hにするには1FFhの後，桁上がりさせる回路が必要となるが今回は省略した．

バンク切り替え

CPUから見たときはまったく同じアドレスでありながら，I/Oポートなどの設定データを併用することでハード的にはまったく別々の領域へのアクセスにする手法．領域(バンク)を切り替えて使うことから「バンク切り替え」という．

ハンドシェイク

データ線のほかに一方からの転送要求信号と，もう一方からの応答信号を使ってデータ転送を行う方式．SCSIバスでも利用されている．相手の応答を見て動作するため確実ではあるものの，伝送速度の点ではやや不利．

ベンダリクエスト

USB機器において，各ベンダが自由に定義して利用できるリクエスト(コマンド)．USBではこのほかにもすべての機器が備えるべき標準リクエスト，共通化作業で決まったクラスリクエストがある．

USBホストコントローラの概要とプロトコルスタックの移植

DPS(Direct Printer Service)

キヤノン，富士写真フイルム，Hewlett-Packard(HP)，オリンパス光学工業，セイコーエプソン，ソニーの6社が策定した，PCを介さずにディジタルカメラの画像を直接プリンタから印刷できる規格(通常，USBではホストコントローラが必要なため，ディジタルカメラからプリンタへの印刷は，必ずUSBホストコントローラを搭載したPCが介在する必要がある)．画像入力デバイスと画像出力デバイスとを，PCを介さずに直接接続し，印刷するためのアプリケーションレベルのインターフェース規格となっている．

FlexiStack

フィリップスセミコンダクターズが発売している，組み込み用USBホストスタックの製品名．FlexiStackは，USBホストスタック，USBデバイススタックおよび手軽なクラスドライバ群により構成されている．μITRON，pSOS，VxWork，DOSおよびLinuxなどの汎用リアルタイムOSのほとんどをサポートしている．国内では，(株)スティルが販売代理店となっている．**図5**に，リアルタイムOS上のFlexiStackの論理構成図を示す．

ISP1362

フィリップスが発売している，USB OTG対応のホストコントローラチップの名称．組み込みシステムやPC周辺機器のポイントツーポイント接続を実現する，1チップOTGホスト/ペリフェラルコントローラ．USB2.0規格(フルスピード，ロースピード)とOTG追加規格Rev.1.0に準拠している．

OHCI (Open Host Controller Interface)

USBホストコントローラの規格の一つ．Intel以外のUSBホストコントローラの多くは，この規格に基づいている．UHCIと比較すると，ソフトウェアの負荷が小さいのが特徴となっている．

〔図5〕リアルタイムOS上のFlexiStack論理構成図

SH7727

日立製作所が開発した32ビットマイクロプロセッサの一つ．SHシリーズの一つで，組み込み機器向けにさまざまな最適化がはかられており，USBファンクションコントローラや，USBホストコントローラを内蔵している．

SolutionEngine

日立製作所の32ビットマイクロプロセッサ“SuperH”を使った，組み込みのユーザーシステムが簡単に構築できるリファレンスプラットホーム．

UHCI (Universal Host Controller Interface)

Intelが主導して策定されたUSBホストコントローラの規格．IntelやVIAのチップセットで採用されている．OHCIと比較して，コストがかからないという特徴がある．

USB Implementers Forum

USBの仕様策定管理団体．USBのコアの仕様や各デバイスクラスの定義仕様書などを行っている．

USB On-The-Go

USB Inplementers Forumによって策定された規格で，パソコンを介することなくUSB機器を相互に接続することができる．「USB OTG」あるいは「USB“On the Go”」とも表記される．現在のところ，転送速度は，ロースピード(1.5Mbps)やフルスピード(12Mbps)のみの対応で，USB2.0で採用されたハイスピード(480Mbps)は，Optional扱いになっている．

USBホストドライバ

USBホストコントローラ用のドライバ．USBホストコントローラには，OHCI，UHCIといった仕様がある．USBを使用するためには，それぞれのホストコントローラに合わせたドライバが必要になる．このドライバをUSBホストドライバという．Windowsでは，UHCIとOHCIのドライバは標準で用意されている．

μITRON

TRONのサブプロジェクトの一つ．組み込み機器用のリアルタイムOSの仕様．日本ではかなりの導入実績がある．

● クライアントドライバ

USBのクライアントデバイス用のドライバ．USBで接続される，各機器ごとに使用されるドライバ．

● ホストコントローラドライバインターフェース

USBプロトコルスタックにおける，ホストコントローラとバスドライバ間のインターフェース仕様．

● マスストレージデバイス

FDD，リムーバブルディスク，メモリカードリーダなどの外部記憶デバイス．

● ミニAコネクタ

USBのホスト側のミニコネクタ．

● ミニBコネクタ

USBのクライアント側のミニコネクタ．

● ミニABコネクタ

USB2.0 OTGで採用されたコネクタ．Aコネクタと Bコネクタの兼用となっており，接続先がAコネクタであれば，接続した側がホストになる．

USB機器開発における USBアナライザの活用法

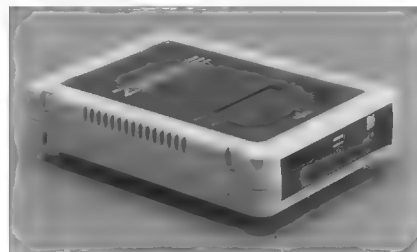
● Endpoint Dataファイル

USBアナライザ「USB ZERONE」のもつEndpointデータ抽出機能で作成されるファイル．USBではデータ転送時にUSBプロトコルに沿ってパケット（TOKEN，HANDSHAKE）が付加され，さらにフラグメント（分割）される場合がある．それら実データ以外を取り除き実データを復元することで転送前のデータとの比較が可能となる．

● Enumerationシーケンス

USBデバイスがUSBバス上に接続された際に，USBホストにより接続されたUSBデバイスの情報を取得するシーケンス．コントロール転送を使用してUSBデバイスのディスクリプタ（Descriptor）データを取得する．

● Frame Analyzer機能

USBにおける単位時間であるフレーム（マイクロフレーム）ごとにデータ転送の状況をグラフ化する機能（**図6**）．データ転送時のギャップやUSBデバイス（Function）のデータ転送パフォーマンスを，視覚的に見ることができる．

〔図6〕Frame Analyzer画面

● Non-Node機能

通常ターゲット機器が接続されたバス上に挿入するバス（プロトコル）アナライザが，測定しているバス上にバス機器（USBであればUSBデバイス，1394であれば1394 Node）として認識されないようにする機能．接続機器の構成（ツリー）に影響を与えないため，特定の機器構成で発生する事象を解析する場合に威力を発揮する．

● QuickTrace

トリガなどの設定なしにUSBバス上のデータをキャプチャする機能．QuickTraceはUSB ZERONEの機能名称であり，USB ZERONE以外のアナライザでは異なる名称で呼ばれている．

● SSTD (Single Step Transaction Debugger)

USBデバイスの評価用ツールで，PC上から1ステップ任意のトランザクションを発生させることができる．USB IFのWebサイト（http://www.usb.org/）で公開されている．

● USB ZERONE

富士通デバイス社製国産USBアナライザ（**写真2**）．IEEE1394アナライザZERONEとともにプロトコルアナライザZERONEファミリの一つ．

〔写真2〕USB ZERONE

● アイソクロナス（Isochronous）転送

帯域の保証された転送方式．Audio/Videoデータなどの転送に用いられる．一定周期でデータを転送できるが，データ内容の保証はない．USBではこのほかにコントロール（Control）転送，バルク（Bulk）転送，インタラプト（Interrupt）転送と，全部で4種の転送方式がある．

● アプリケーション層よりの解析

アナライザを用いてプロトコルなどソフトウェアのデバッグを目的とした解析を行うこと．

● キャプチャ

USBアナライザにおいて，USBバス上に流れるデータを記録すること．USBアナライザの役割は，キャプチャして効率よく解析（トレース）する手助けをすることである．

〔図7〕翻訳画面

Analyzer

T	C	M	Packet...	E	S	S...	PID	Packet name	A...	E...	DATA	Time
			000000945		FS	80	DATA0	GET DESC [CFG]	01	0	8	05s085ms148us916ns
						0000:		80 06 00 02 00 00 20 00			....	
			000000946		FS	80	ACK					05s085ms157us350ns
			000000948		FS	80	IN		01	0		05s086ms145us750ns
			000000949		FS	80	DATA1	Configuration descriptor	01	0	32	05s086ms148us950ns
						0000:		09 02 20 00 01 01 03 80-96 09 04 00 00 02 08 06			....	
						0010:		50 00 07 05 01 02 40 00-00 07 05 82 02 40 00 00			P.....@.......@..	
			000000950		FS	80	ACK					05s086ms173us750ns
			000000952		FS	80	OUT		01	0		05s087ms145us683ns
			000000953		FS	80	DATA1		01	0	0	05s087ms148us766ns
			000000954		FS	80	ACK					05s087ms151us900ns
			000000957		FS	80	SETUP		01	0		05s089ms145us533ns
			000000958		FS	80	DATA0	SET CFG	01	0	8	05s089ms148us616ns
						0000:		00 09 01 00 00 00 00 00			....	
			000000959		FS	80	ACK					05s089ms157us066ns
			000000961		FS	80	IN		01	0		05s090ms145us450ns
			000000962		FS	80	DATA1		01	0	0	05s090ms148us650ns
			000000963		FS	80	ACK					05s090ms152us033ns
			000001042		FS	80	SETUP		01	0		05s168ms139us650ns
			000001043		FS	80	DATA0	MAX_LUN [Storage]	01	0	8	05s168ms142us733ns
						0000:		A1 FE 00 00 00 00 01 00			....	
			000001044		FS	80	ACK					05s168ms151us250ns
			000001046		FS	80	IN		01	0		05s169ms139us566ns
			000001047		FS	80	DATA1		01	0	1	05s169ms142us750ns
						0000:		00			.	
			000001048		FS	80	ACK					05s169ms146us900ns
			000001050		FS	80	OUT		01	0		05s170ms139us500ns
			000001051		FS	80	DATA1		01	0	0	05s170ms142us583ns
			000001052		FS	80	ACK					05s170ms145us683ns
			000001055		FS	80	OUT		01	1		05s172ms139us350ns
			000001056		FS	80	DATA0	Storage [CBW]	01	1	31	05s172ms142us433ns
						0000:		55 53 42 43 80 48 A8 81-24 00 00 00 80 00 06 12			USBC.H..$.......	
						0010:		00 00 00 24 00 00 00 00-00 00 00 00 00 00 00			...$...........	
			000001057		FS	80	ACK					05s172ms166us200ns
			000001060		FS	80	IN		01	2		05s174ms139us200ns
			000001061		FS	80	DATA0		01	2	36	05s174ms142us400ns
			000001062		FS	80	ACK					05s174ms169us783ns
			000001065		FS	80	IN		01	2		05s176ms139us050ns
			000001066		FS	80	DATA1	Storage [CSW]	01	2	13	05s176ms142us233ns
						0000:		55 53 42 53 80 48 A8 81-00 00 00 00 00			USBS.H.......	
			000001067		FS	80	ACK					05s176ms154us300ns
			000001070		FS	80	OUT		01	1		05s178ms138us900ns

トレース

記録したデータ(キャプチャデータ)を文字どおりなぞる(トレース)こと．アナライザの機能を用いてデータを解析することを指す．

パケットジェネレータ

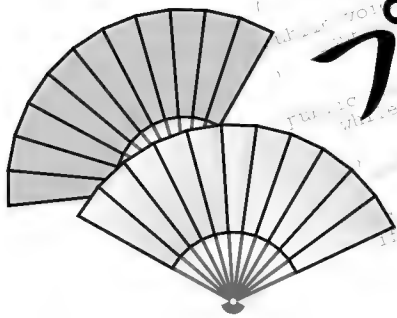
USBバス上に任意のUSBパケットを送出できる開発ツール．PCを利用してソフトウェアで実現するものと，専用ハードウェアをもつものがある．

物理層よりの解析

アナライザを用いて電気的事象を解析すること．ハードウェアのデバッグを目的とし，オシロスコープやロジックアナライザと似た解析を行うこと．

翻訳機能

キャプチャしたデータ(バイナリデータ)をUSB規格や上位規格(Device Classなど)に沿って翻訳表示する機能(**図7**)．USBアナライザでは，パケットレベル，トランザクションレベル，トランスファレベルなど，階層ごとの翻訳機能が求められる．

桑野雅彦　パステルマジック
芹井滋喜　(株)ソリトンウェーブ
谷本和俊　富士通デバイス(株)

宮坂電人

第5回 継承禁止令

まったくナンセンスとしか思えないような規約を採用している現場を見かけることがあります．たとえば，COBOLプログラマ出身者が多い職場で，「ローカル変数使用禁止」とか(昔のCOBOLではグローバル変数のみでローカル変数がなく，ローカル変数が何であるかを理解していないプログラマが珍しくなかったようだ)，「関数の頭は小文字で始めて，変数の頭を大文字で始める」とか(そうしないと関数と変数の区別がつかないという主張らしい)．

こうした「妙な」規約が出てくる背景として，それまで自分たちがなじんできたものと相容れないものを警戒するという感情的な原因があったり，そういう規約を採用しないと落とし穴にハマるという主張があります．しかし第三者には，なぜ警戒するのか，なぜ落とし穴にハマるのか理解しにくいものがあります．

最近ではオブジェクト指向開発が本格化してきたせいか，それがらみの変な規約を見かけることがあります．たとえば，「継承禁止令」というのがそうです．これは言葉どおりの意味で，継承を使うなということです．しかし，クラスライブラリやフレームワークを利用すると，どうしても継承は避けられません．しかも，継承して使うことが前提のクラスがあるので，規約を守るためにはどうすればよいのだろうと筆者は思うのですが，当事者はいたってマジメなので困惑をおぼえます．

継承禁止令を主張する人たちも，やはりローカル変数の使用禁止を主張する人たちと同様，自分たちがなじんできたものと相容れないものを警戒する感情面が大きいのですが，話を聞くと，それも止むを得ないかと思うこともあります．しかし，よくよく考えると継承が何たるかを把握していないため，落とし穴にハマりこんでいるだけにすぎないという面もあります．

今回は，継承のもつやっかいな性質と，それを回避するための原則について検討しましょう．

継承は特殊化か拡張か

継承禁止令を守っていくと，似たような働きのあるクラスがすでにあるとわかっていながら，新たにクラスを一から作る"車輪の再発明"的なジレンマに苦しめられるハメになります．そのジレンマを克服しても，結局は既存クラスのソースを"コピー＆ペースト"するという指の運動をすることになります．

継承は，こういった空しい作業を減らす"差分プログラミング"の手段であることはすでによく知られていることです．本連載の第3回でも述べたとおり，継承を利用することで一から作成せずとも，ベースとなるクラスがあるなら機能拡張を少ない労力でできるメリットがあります．つまり継承とは，"楽に機能拡張するしくみ"です．

ところで，次のような例ではどうでしょうか？　長方形を表現するRectClassがあったとします(**リスト1**)．さらに，それを継承した正方形を表現するSquareClassがあったとします(**リスト2**)．

正方形の場合，縦幅と横幅は等しいので，setSizeの引き数は一つだけで十分です．これで一見問題がないように思えますが，**リスト3**のようなプログラムではどうでしょうか？　問題が起きるのは「aSquare.setSize(10,20)」の行です．ここでRectClassのsetSizeを呼んでしまい，結果的に縦幅と横幅が一致しない，つまり正方形ではない設定が行われます．ここで起きたトラブルは，呼ばれると都合が悪い継承元のメソッドを継承したクラスでオーバライドしていなかったのが原因です．わかってしまえばじつにバカらしいトラブルですが，

〔リスト1〕長方形のクラスRectClass

```
public class RectClass {                    //長方形のクラス
    protected double mWidth,mHeight;
    public void setSize(double iWidth,double iHeight) { //横幅と縦幅を変更する
        mWidth = iWidth;
        mHeight = iHeight;
    }
    ...(略)...
}
```

〔リスト2〕正方形のクラスSquareClass

```
public class SquareClass extends RectClass {
    public void setSize(double iWidth) {
        mWidth = mHeight = iWidth;
    }
}
```

〔リスト3〕継承の盲点をついたプログラム(1)

```
public void test() {
    RectClass aRect = new RectClass();
    aRect.setSize(10,20);

    SquareClass aSquare = new SquareClass();
    aSquare.setSize(30);
    aSquare.setSize(10,20); //これができていいのだろうか？
}
```

〔リスト4〕SquareClassの修正

```
public class SquareClass extends RectClass {
    ...(略)...
    public void setSize(double iWidth,double iHeight) {
        setSize(iWidth);
    }
}
```

〔リスト5〕継承の盲点をついたプログラム(2)

```
public void test2() {
    double aWidth = 10;
    double aHeight = 20;

    SquareClass aSquare = new SquareClass();
    aSquare.setSize(aWidth,aHeight);

    double aArea = aWidth * aHeight;
    ...以下，aAreaがaSquareの面積である前提で処理が進む
}
```

それだけにハマってしまうと，まさかそんな単純なミスがあるとは気づかず頭をかかえることになります．

いうまでもなく，これを対策するのはSquareClassの中で「public void setSize(double iWidth, double iHeight)」をオーバライドするだけです．たとえば，**リスト4**のようにすればよいでしょう．

しかし，この対策をほどこしても，**リスト5**のようなケースではどうなるでしょうか？

これは，さきほどのトラブルとは違う種類のトラブルです．いうまでもなくaSquareの面積は10 × 10 = 100であるはずなのに，test2内では10 × 20 = 200であるという勘違いで処理が進むわけです．じつは，こういったトラブルは継承の性質を根本的に理解していないために起こり，それが明確に認識されず曖昧にされるため「継承禁止令」のような対症療法でごまかされる結果にもなり得るわけです．

継承は“機能拡張”であり，“差分プログラミング”の手段ですが，ここで示した「長方形→正方形のジレンマ(楕円→円のジレンマで説明している例もある)」のような，どこが“拡張”なんだという例もときどきあります．つまり，柔軟性や可能性が狭まる“特殊化”(生物学に詳しい人なら“進化の袋小路”という表現のほうがわかりやすいかもしれない)ではないのかと．

じつは，本講座の第2回で紹介した『オブジェクト指向入門[注1]』でも，クラスを“型”とみるか“モジュール”とみるかで，とらえ方が変わり，

- 型とみる → 特殊化
- モジュールとみる → 拡張

になるとの記述があります．クラスをとらえる場合，どうしても“メタファ(暗喩)”でとらえ，その実態を曖昧にとらえ誤謬(ごびゅう)に陥りやすくなるオブジェクト指向ならではの落とし穴ともいえます．いずれにせよ，ここで起きた問題の正体をもう少し追求してみましょう．

私作る人，あなた使う人

昔「私作る人，あなた食べる人」というテレビCMがあり，これは女性蔑視ではないかと物議をかもしたものです．ところで，これと継承がどうかかわるのだと疑問に思われる読者もおられるに違いないので説明しておきましょう．

さきほどのトラブルはあまりにもわざとらしい例で，こんなことは起きるはずがないと思われる読者も多数おられることでしょう．たしかに，こんなに“わかりやすい”例はあまりありません．たいていの場合，RectClassとSquareClassを作る人は同一人物ですし，しかも小規模なので，ほとんどは「私クラスを作る人，使うのも私」でしょう．もう少し規模が大きくなっても「私クラスを作る人，あなた使う人」です．

ところが，さらに規模が大きくなったりクラスライブラリやフレームワークを使いだすと「私ベースクラスを作る人，あなた継承する人」という事態が起こり得ます．そうなると，どういう愉快な(？)事態が起きるかといえば，「あなたベースクラスの中身をさっぱりわからず継承する人」が出てくるわけです．

すでにクラスライブラリやフレームワークで頭をかかえた経験のある人ならピンと来るでしょうが，あるメソッドが用意されていた場合，“継承”という面で次の2パターンに分類できます．

(1) 外部から，そのメソッドを明示的に呼び出す
(2) 継承したクラスで，そのメソッドをオーバライドする(呼び出す主体は外部ではなくライブラリやフレームワーク)

伝統的なプログラミングでは(1)がすべてですが，オブジェクト指向開発だと(2)のパターンがあり，しかも親クラスのメソッドをどう使うかで頭をかかえてしまう場合があります．つまり，あるメソッドをオーバライドするとき，

(A) 親クラスのメソッドを呼び出してはいけない
(B) 親クラスのメソッドを呼び出さねばならない

の2パターンがあるわけです．さらに(B)も，

(B-1) 親クラスのメソッドを任意場所で呼ぶ
(B-2) 親クラスのメソッドを先頭で呼ぶ
(B-3) 親クラスのメソッドを最後に呼ぶ

というバリエーションがあり，呼び方を間違えると不可解なト

注1：バートランド・メイヤー著，(株)アスキー，ISBN4-7561-0050-3．原題は*Object-Oriented Software Construction*．翻訳書は1990年発行で初版だが，原書はすでにSecond Editionが出ている．そちらはPrentice Hall，ISBN 0-13-629155-4．参考URLは http://archive.eiffel.com/doc/oosc/

ラブルで頭をかかえるのはいうまでもありません．

じつをいうと，筆者は未だにクラスライブラリやフレームワークを使うとき，どのように親クラスのメソッドを呼べばよいのか（あるいは呼んではいけないのか）で頭をかかえることがあります．たしかオブジェクト指向では「オブジェクトやクラスの詳細を気にせずブラックボックス化できる」という“情報隠蔽”の御利益がうたわれていたはずなのに，気がつくと，どのように親クラスのメソッドを呼ぶべきかを「クラスライブラリのソースを解析」して検討している自分の姿に苦笑するわけです〔余談だが，デバッガを使ってクラスライブラリのソースをトレースするとわかりやすい．デバッガは，バグ追跡だけでなく他人のソースを解析するツールとしても重宝する（笑）〕．

ちなみに，親クラスの中身を理解しないと継承できないのか，理解しなくても継承できるのかで，

- ブラックボックスな継承 → あるクラスを継承しようとしたとき，その実装を理解しなくても継承できる
- ホワイトボックスな継承 → あるクラスを継承しようとしたとき，その実装を理解しないと継承できない

という分類もあるそうです．

Liskovの置換原則（Liskov Substitution Principle）

いずれにせよ継承がもつ問題点（？）として，

- あるクラスを継承しようとすると，そのクラスの“実装”を理解しないといけない場合がある
- あるクラスを継承した別のクラスを利用するとき，両方のクラスの“実装”を理解しないといけない場合がある

というわけです．最初に出てきたSquareClassでsetSize(double iWidth, double iHeight)のオーバライドを忘れたために起きたトラブルは，まさにこれが原因でしょう．

では，二つ目のトラブル「面積を勘違いした」はどうでしょうか．これは継承する側の問題ではなく，クラスを利用する側の問題です．利用する側がクラスの実装を知りすぎたという場合もあるでしょうが，クラスの“実装を自分なりに想像あるいは期待”していたのが根本的な原因です．じつは，この種のトラブルを予防するための原則として，“Liskov[注2]の置換原則”というものがあります（参考URL：http://www.objectmentor.com/resources/articles/lsp.pdf）．それは，

> 『親クラスへのポインタまたは参照を利用するファンクションは子クラスのオブジェクトをその中身を知ることなく利用できねばならない』（原文は「Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.」）

というものです．これだけだと何のことやらよくわかりません．そこで次のような例題で，この原則の意味を理解していきましょう．

- 賛成票と反対票を集計するクラスを作り，賛成の比率を求めるプログラムを作る

賛成の比率を求めるプログラム

まず，賛成票と反対票を集計するクラスは，**リスト6**のようになります．次に，賛成の比率を求めるプログラムは**リスト7**のようになります．

ここでは賛成票が40，反対票が10で賛成比率は80%になり，どうやらうまくいっているかのように思えます．ところが，次のようなクラスを継承で作り出されると，とたんに破綻します．

- 賛成票と反対票以外に無効票も集計するクラス

〔リスト6〕AOClass

```
public class AOClass {
    protected int mApprove  = 0;        //賛成票
    protected int mOpposite = 0;        //反対票

    public void addApprove(int iD) {    //賛成票を増やす
        mApprove += iD;
    }

    public void addOpposite(int iD) {   //反対票を増やす
        mOpposite += iD;
    }

    public int getApprove() {           //賛成票の数をえる
        return mApprove;
    }

    public int getOpposite() {          //反対票の数をえる
        return mOpposite;
    }
    ...(略)...
}
```

〔リスト7〕賛成の比率を求めるプログラム（1）

```
public double getApproveRate(AOClass iA) {
    double aA = (double)(iA.getApprove());
    double aB = (double)(iA.getApprove() + iA.getOpposite());
    return aA / aB * 100.0;
}

public void testAOClass() {
    AOClass a1 = new AOClass();
    a1.addApprove(40);
    a1.addOpposite(10);
    System.out.println("Approve rate = " + getApproveRate(a1) + " %"); // 80%
}
```

注2：Barbara Liskov，CLUというプログラミング言語の開発にかかわったことで有名．業績はhttp://www.acm.org/fcrc/plenary.htm#Liskovを参照．

これは**リスト8**のようになります．そして，賛成の比率を求めるプログラムを使うと(**リスト9**)，賛成比率が40%にならないといけないのに80%になるわけです．ここまで来るともうおわかりでしょうが，getApproveRateメソッドで全体の数を求めるのに，getApproveとgetOppositeの合計を使うという「実装を自分なりに想像あるいは期待」した落とし穴にハマったわけです．

再び，Liskovの置換原則

さて，ここでLiskovの置換原則を使い，testAOClassという“ファンクション”の問題点をはっきりさせましょう．つまり，「親クラス(AOClassのこと)への参照を利用するファンクション(testAOClassのこと)は，子クラス(AOandIClassのこと)のオブジェクトをその中身を知ることなく利用」できていません．それ以前に破綻しています．また，**リスト10**のような解決案(？)はどうでしょうか．これは計算結果は正しく出そうですが，問題の根本的な解決にはなっていません．これだと「子クラスのオブジェクトをその中身を知ることなく利用」できていないので，完全にLiskovの置換原則に反しています．

ここで示した例は，多少わざとらしい点があるように見受けられますが，実際の開発現場では，こういった「クラス判定による分岐」のたぐいの“対症療法”が意外と横行しています．対症療法ですから，たとえばAOandIClassを継承した別クラスが作成されると，この方法ではたちまち例外が出てしまい「分岐の増設」を要求されるわけです．そして，こんな要求が延々と続こうものなら「継承禁止令」が発布される可能性が高まるわけです．

いうまでもないことですが，根本的な解決とは全体の数を求めるメソッドを親クラスの段階で装備することであり，小手先の対症療法でしのぐことではありません．

〔リスト8〕AOandIClass

```
public class AOandIClass extends AOClass {
    protected int mInvalid  = 0;        //無効票

    public void addInvalid(int iD) {    //無効票を増やす
        mInvalid += iD;
    }

    public int getInvalid() {           //無効票の数をえる
        return mInvalid;
    }
    ...(略)...
}
```

〔リスト9〕賛成の比率を求めるプログラム(2)

```
public void testAOClass() {
    AOandIClass a2 = new AOandIClass();
    a2.addApprove(40);
    a2.addOpposite(10);
    a2.addInvalid(50);
    System.out.println("Approve rate = " +
        getApproveRate(a2) + " %"); // 80% (?)
}
```

〔リスト10〕賛成の比率を求めるプログラム(3)

```
public double getApproveRateEx(AOClass iA) throws Exception
{
    double aA = (double)(iA.getApprove());
    double aB;
    String aClassName = iA.getClass().getName();
    if(aClassName.equals("AOClass")){
        aB = (double)(iA.getApprove() + iA.getOpposite());
    }else if(aClassName.equals("AOandIClass")){
        AOandIClass aO = (AOandIClass)iA;
        aB = (double)(aO.getApprove() + aO.getOpposite()
                                      + aO.getInvalid());
    }else{
        throw new Exception("unknown class : " + aClassName);
    }
    return aA / aB * 100.0;
}
```

依存逆転の原則(Dependency Inversion Principle)

Liskovの置換原則は，ある種の“べからず集”や“禁じ手”とも考えられます．残念ながら，筆者が経験するところでは「これをするな」といわれても，素直に意見を聞くより反発されることが多いようです．また，具体的にこうしようという方向に向いていないのも弱点です．ダメというなら，どうすりゃいいんだ，と逆ギレされるとお手上げです．

そこで，さらに具体性が高く，方向が定まっている原則として“依存逆転の原則”というものがあります(参考URL：http://www.objectmentor.com/resources/articles/dip.pdf)．これは，

(A)『高レベルのモジュールは下位レベルのモジュールに依存すべきでない．両方のモジュールは抽象化したものに依存すべきである』(原文は，「High level modules should not depend upon low level modules. Both should depend upon abstractions.」)

(B)『抽象化したものは詳細なものに依存すべきでない．詳細なものは抽象化したものに依存すべきである』(原文は，「Abstractions should not depend upon details. Details should depend upon abstractions.」)

というものです．しかし，この文章だけではさっぱりわかりません．第一，「抽象化したもの(原文のAbstractionには「抽出/分離」の意味もある)」とは何なのか意味不明です．また，「逆転」とは何と何が逆転しているのでしょうか．そこで，ここでも例題を作ってみましょう．

- あるファイルから読み取ったデータを圧縮し，別のファイルに書き込む
- さらにファイルだけでなくネットワークや，その他のI/Oにも対応できるようにする

単純な圧縮プログラム

あるファイルから読み取ったデータを圧縮し，別のファイル

〔リスト11〕**MyCompress**(最初の実装)

```
public class MyCompress {
    ...(略)...
    protected NSData compressMain(NSData iData) {                //データオブジェクトを圧縮する
        ...(略)...
    }

    public int fileToFile(MyFileReader iFR,MyFileWriter oFW) {   //ファイル圧縮
        int aRet;
        if(iFR.setUp() != 0){                                    //ファイルからの読み取り準備をする
            aRet = ReadErr;
        }else{
            if(oFW.setUp() != 0){                                //ファイルへの書き込み準備をする
                aRet = WriteErr;
            }else{
                NSData aReadData = iFR.readAllData();            //ファイルからすべて読み取る
                if(aReadData == null){
                    aRet = ReadErr;
                }else{
                    NSData aCompData = compressMain(aReadData);  //圧縮する
                    if(aCompData == null){
                        aRet = CompressErr;
                    }else{
                        if(oFW.writeAllData(aCompData) != 0){    //ファイルに圧縮データをすべて書き込む
                            aRet = WriteErr;
                        }else{
                            aRet = OK;
                        }
                    }
                }
                oFW.cleanUp();                                   //後始末処理
            }
            iFR.cleanUp();                                       //後始末処理
        }
        return aRet;
    }
}
```

に書き込む例を示すと**リスト11**のようになります．

MyFileReaderはファイルを読み込むクラス，MyFileWriterはファイルに書き込むクラスで，この二つは以下のようなメソッドをもっています．

- int setUp()――初期化処理，戻り値が0ならOK，0以外はエラー
- int writeAllData(NSData iData)――データオブジェクトをファイルに書き込む．戻り値が0ならOK，0以外はエラー
- NSData readAllData()――ファイルを読み取り，データオブジェクトにする
- void cleanUp()――後始末処理

圧縮プログラムを拡張する

次に，ネットワークからデータを読み込むMyNetReader，ネットワークにデータを書き込むMyNetWriterを用意し，ファイルの読み書きと同様，ネットワークの読み書きも対応させます．MyNetReaderのメソッドの外部仕様はMyFileReaderとほぼ同じで，MyNetWriterのそれもMyFileWriterとほぼ同じとします．この場合，読み書きのパターンは2×2＝4通りになります．つまり，

- int netToNet(MyNetReader iFR, MyNetWriter oFW)――ネットから読んで圧縮したものをネットに書く
- int fileToNet(MyFileReader iFR, MyNetWriter oFW)――ファイルから読んで圧縮したものをネットに書く
- int netToFile(MyNetReader iFR, MyFileWriter oFW)――ネットから読んで圧縮したものをファイルに書く

という三つのメソッドを余分に用意しないといけません．しかし，こんなやり方をすると，I/Oが増設されるたびに読み書きのパターンが増え，組み合わせの爆発現象が起き，メソッドの数も爆発します．一つのメソッドですべてのI/Oをサポートするようにすればよいのですが，だからといって**リスト12**のようなやり方は根本的な解決になっていません．

これでは，「親クラス(Objectのこと)への参照を利用するファンクション(anyToAnyのこと)は子クラス(MyFileReader，MyNetReader，MyFileWriter，MyNetWriterのこと)のオブジェクトをその中身を知ることなく利用」できていないので，さきほど述べたLiskovの置換原則に反しています．もちろん，新たにI/Oが出現すると，そのたびにメソッドを変更しないといけないので手間がかかります．

Template Methodパターンによる対応

残念なのは，せっかくファイルの読み書きもネットの読み書きも，どういうメソッドを用意すべきか“抽象化”できているのに，それをまったく活用していない点です．つまり，読み込み側は**リスト13**のように，書き込み側は**リスト14**のように抽象

〔リスト12〕**MyCompress**（ネットワークにも対応）

```
public class MyCompress {
    ...(略)...
    private int aTA_R_Setup(String iRN,Object iOR) {
        if(iRN.equals("MyFileReader")){
            return ((MyFileReader)iOR).setUp() == 0 ? OK : ReadErr;
        }else if(iRN.equals("MyNetReader")){
            return ((MyNetReader)iOR).setUp() == 0 ? OK : ReadErr;
        }
        return ReadErr;
    }

    private int aTA_W_Setup(String iWN,Object iOW) {
        if(iWN.equals("MyFileWriter")){
            return ((MyFileWriter)iOW).setUp() == 0 ? OK : WriteErr;
        }else if(iWN.equals("MyNetWriter")){
            return ((MyNetWriter)iOW).setUp() == 0 ? OK : WriteErr;
        }
        return WriteErr;
    }

    private NSData aTA_ReadAllData(String iRN,Object iOR) {
        if(iRN.equals("MyFileReader")){
            return ((MyFileReader)iOR).readAllData();
        }else if(iRN.equals("MyNetReader")){
            return ((MyNetReader)iOR).readAllData();
        }
        return null;
    }

    private int aTA_WriteAllData(NSData iData,String iWN,Object iOW) {
        if(iWN.equals("MyFileWriter")){
            return ((MyFileWriter)iOW).writeAllData(iData) == 0 ? OK : WriteErr;
        }else if(iWN.equals("MyNetWriter")){
            return ((MyNetWriter)iOW).writeAllData(iData) == 0 ? OK : WriteErr;
        }
        return WriteErr;
    }

    private void aTA_W_CleanUp(String iWN,Object iOW) {
        if(iWN.equals("MyFileWriter")){
            ((MyFileWriter)iOW).cleanUp();
        }else if(iWN.equals("MyNetWriter")){
            ((MyNetWriter)iOW).cleanUp();
        }
    }

    private void aTA_R_CleanUp(String iRN,Object iOR) {
        if(iRN.equals("MyFileReader")){
            ((MyFileReader)iOR).cleanUp();
        }else if(iRN.equals("MyNetReader")){
            ((MyNetReader)iOR).cleanUp();
        }
    }

    public int anyToAny(Object iOR,Object iOW) {                                    //汎用圧縮（?）
        String aReadClassName = iOR.getClass().getName();
        String aWriteClassName = iOW.getClass().getName();
        //System.out.println("aReadClassName:" + aReadClassName + ",aWriteClassName:" + aWriteClassName);
        int aRet = aTA_R_Setup(aReadClassName,iOR);                                 //読み取り側の準備
        if(aRet == OK){
            aRet = aTA_W_Setup(aWriteClassName,iOW);                                //書き込み側の準備
            if(aRet == OK){
                NSData aReadData = aTA_ReadAllData(aReadClassName,iOR);             //データをすべて読み取る
                if(aReadData == null){
                    aRet = ReadErr;
                }else{
                    NSData aCompData = compressMain(aReadData);                     //圧縮する
                    if(aCompData == null){
                        aRet = CompressErr;
                    }else{
                        aRet = aTA_WriteAllData(aCompData,aWriteClassName,iOW);     //圧縮データを書き込む
                    }
                }
                aTA_W_CleanUp(aWriteClassName,iOW);                                 //書き込み側の後始末
            }
            aTA_R_CleanUp(aReadClassName,iOR);                                      //読み取り側の後始末
        }
        return aRet;
    }
}
```

〔リスト13〕**MyAnyReader**

```
public interface MyAnyReader {
    public int setUp();             //初期化処理，戻り値が0ならOK，0以外はエラー
    public NSData readAllData();    //読み取ったものをデータオブジェクトにする，エラーならnullを返す
    public void cleanUp();          //後始末処理
}
```

〔リスト14〕**MyAnyWriter**

```
public interface MyAnyWriter {
    public int setUp();                        //初期化処理，戻り値が0ならOK，0以外はエラー
    public int writeAllData(NSData iData);     //データオブジェクトを書き込む，戻り値が0ならOK，0以外はエラー
    public void cleanUp();                     //後始末処理
}
```

〔リスト15〕**MyCompress**（最終案）

```
public class MyCompress {
    ...(略)...
    public int anyToAny(MyAnyReader iAR,MyAnyWriter oAW) {      //汎用圧縮
        int aRet;
        if(iAR.setUp() != 0){                                   //読み取り準備をする
            aRet = ReadErr;
        }else{
            if(oAW.setUp() != 0){                               //書き込み準備をする
                aRet = WriteErr;
            }else{
                NSData aReadData = iAR.readAllData();           //すべて読み取る
                if(aReadData == null){
                    aRet = ReadErr;
                }else{
                    NSData aCompData = compressMain(aReadData);  //圧縮する
                    if(aCompData == null){
                        aRet = CompressErr;
                    }else{
                        if(oAW.writeAllData(aCompData) != 0){   //圧縮データをすべて書き込む
                            aRet = WriteErr;
                        }else{
                            aRet = OK;
                        }
                    }
                }
                oAW.cleanUp();                                  //後始末処理
            }
            iAR.cleanUp();                                      //後始末処理
        }
        return aRet;
    }
}
```

化したインターフェースをもっているわけです．

ということは，圧縮プログラムは**リスト15**のように，MyFileReader，MyNetReader，MyFileWriter，MyNetWriterを引き数にするのではなく，抽象化したインターフェースであるMyAnyReader，MyAnyWriterを引き数にすべきということです．そしてMyFileReader，MyNetReaderはMyAnyReaderを継承し，MyFileWriter，MyNetWriterはMyAnyWriterを継承するようにします．また，今後増設されるI/OのクラスもMyAnyReader，MyAnyWriterを継承すれば，anyToAnyメソッドの中身は書き換える頻度がかなり減るはずです．

抽象化

抽象化というと，まるで“難しいこと”，“曖昧なこと”と誤解されるのですが，実際はその逆の“簡素化”，“明確化”へ導く性質をもっています．そうなっていないとすれば，たぶんいままで抽象化と思い込んでいたものは何か別物だったとしかいいようがありません．

次回も“依存逆転の原則”を検討し，さらに実際のプログラミングの現場ではびこる“Bad Design”について考察したいと思います．

みやさか・でんと　miyadent@anet.ne.jp

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第⑳回　ストリング命令とシステム命令の概要　大貫広幸

今回は，x86系CPUの汎用命令のうち，最後に残ったストリング命令の詳細と，CPU自体を制御するためのシステム命令の概要について説明します．

ストリング命令

これまで，x86系CPUの汎用命令について説明してきましたが，今回のストリング命令の説明で，汎用命令についてはすべて説明したことになります．

ストリング命令は，メモリ上にあるバイトストリングを転送，比較，スキャンするものです(**表1**)．ストリング命令は，二つの命令の組み合わせでできています．一つが転送，比較，スキャンといった動作を指定する動作命令です．もう一つが動作を何回繰り返すのかの指定を行う命令です．繰り返し指定命令の使用は任意で，指定がなければ動作命令が1回だけ実行されます．繰り返し指定命令を使用する場合は，必ず動作を指定する命令の前に置く必要があります．

ストリング命令は，メモリ上の転送元と転送先を指定するレジスタを固定しています．転送元(SOU)はDS:(E)SI，転送先(DEST)はES:(E)DIです(**図1**)．

WindowsやLinuxの32ビットプログラムでは，セグメントレジスタDSとESは同じ物理セグメントを示しているので，転送元はESI，転送先はEDIでのみ指定することになります．転送方向はフラグDFにより指定します．DF＝0なら＋方向，DF＝1ならー方向のアドレスとなります．転送元と転送先を示すレジスタ(E)SIと(E)DIは，動作終了後「転送した個数×1データのバイト数」分だけ，DFにしたがって＋あるいはーされます．

動作を指定する命令としては，MOVS，CMPS，SCAS，LODS，

〔表1〕x86系の32ビットCPUのストリング命令

分　類	インストラクション名	動　作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
ストリング命令	MOVS	Move Data from String to String バイトストリングの1データ分の転送をレジスタを使わずメモリ-メモリ間で行う	・	・	・	・	・	・
	CMPS	Compare String Operands バイトストリングの1データ分の比較をレジスタを使わずメモリ-メモリ間で行う	*	*	*	*	*	*
	SCAS	Scan String バイトストリングDESTの1データ分の値とアキュムレータ(AL，AX，EAX)の値を比較する	*	*	*	*	*	*
	LODS	Load String バイトストリングSOUの1データ分の値をアキュムレータ(AL，AX，EAX)にロードする	・	・	・	・	・	・
	STOS	Store String バイトストリングDESTへアキュムレータ(AL，AX，EAX)の値をストアする	・	・	・	・	・	・
	INS	Input from Port to String バイトストリングDESTへレジスタDXが示すI/Oポートから1データ分入力しストアする	・	・	・	・	・	・
	OUTS	Output String to Port バイトストリングSOUの1データ分の値をレジスタDXが示すI/Oポートへ出力する	・	・	・	・	・	・
	REP REPcc	Repeat String Operation Prefix 上記MOVS～OUTSのストリング命令の前に，この命令を付加することでレジスタ(E)CXの回数上記MOVS～OUTSを実行する ccが指定されていたら，繰り返しの途中でccの条件が不成立になったら繰り返しをやめる	・	・	・	・	・	・

注1：表中のDESTはdestination(先)，SOUはsource(元)．　注2：表中の影響を受けるフラグの記号は次の状態を表す．
・＝変化ない　＊＝結果にしたがい変化する

STOS，INS，OUTSの7命令があります．また，繰り返し指定の命令としては，REP，REPE，REPZ，REPNE，REPNZの5命令があります．

このストリング命令のMASMでの記述例を**リスト1**に，gasでの記述例を**リスト2**に示します．

● MOVS命令

バイトストリング1データ（1バイト，1ワード，1ダブルワード）の転送をレジスタを使用しないでメモリ-メモリ間で転送します〔**図2**(**a**)〕．

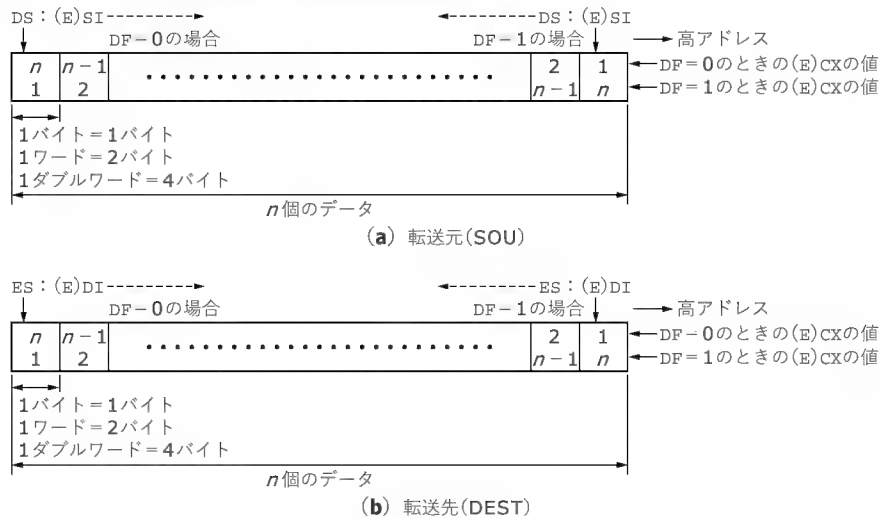
● CMPS命令

バイトストリング1データ（1バイト，1ワード，1ダブルワード）の比較をレジスタを使用しないでメモリ-メモリ間で行います〔**図2**(**b**)〕．

〔図1〕ストリング命令の転送元と転送先の指定

(**a**) 転送元(SOU)

(**b**) 転送先(DEST)

CMPS命令の比較は，他の命令とは異なり記述上，転送元(SOU)と転送先(DEST)のオペランドの指定が逆になります．つまり，MASMの場合は「SOU，DEST」の順でオペランドを記述し，gasの場合は「DEST，SOU」の順でオペランドを記述します．

比較のための減算は，SOU − DESTで行われ，減算結果は捨てられます．そのため，SOUもDESTも変化しませんが，

〔リスト1〕MASMのストリング命令の記述例

```
                              .586
                              .model flat

00000000                      .data
00000000  00000064 [          dtByteS   db    100 dup(1)
           01
          ]
00000064  000000C8 [          dtWordS   dw    200 dup(2)
           0002
          ]
000001F4  0000012C [          dtDWordS  dd    300 dup(4)
           00000004
          ]

00000000                      .data?
00000000  00000064 [          dtByteD   db    100 dup(?)
           00
          ]
00000064  000000C8 [          dtWordD   dw    200 dup(?)
           0000
          ]
000001F4  0000012C [          dtDWordD  dd    300 dup(?)
           00000000
          ]

00000000                      .code
00000000  A4                          movsb
00000001  A4                          movs    es:dtByteD,dtByteS
00000002  A4                          movs    es:dtByteD,[esi]
00000003  A4                          movs    byte ptr es:[edi],dtByteS
00000004  A4                          movs    byte ptr es:[edi],byte ptr [esi]
00000005  A4                          movs    byte ptr es:[edi],[esi]
00000006  66| A5                      movsw
00000008  66| A5                      movs    es:dtWordD,dtWordS
0000000A  66| A5                      movs    es:dtWordD,[esi]
0000000C  66| A5                      movs    word ptr es:[edi],dtWordS
0000000E  66| A5                      movs    word ptr es:[edi],word ptr [esi]
00000010  66| A5                      movs    word ptr es:[edi],[esi]
00000012  A5                          movsd
00000013  A5                          movs    es:dtDWordD,dtDWordS
00000014  A5                          movs    es:dtDWordD,[esi]
00000015  A5                          movs    dword ptr es:[edi],dtDWordS
00000016  A5                          movs    dword ptr es:[edi],dword ptr [esi]
00000017  A5                          movs    dword ptr es:[edi],[esi]
```

オペランドを指定しない場合は，B(バイト)，W(ワード)，D(ダブルワード)で1データのサイズを指定できる

ニモニックにB，W，Dを指定しない場合は，オペランドでDESTとSOUを記述し1データのサイズを指定する．オペランドにはメモリ上の値を示すシンボルか，実際の転送に使われるレジスタを指定する．DEST，SOUの二つともレジスタの場合は「PTR演算子」で1データのサイズを指定する必要がある．このオペランドは，1データのサイズを得るのみに使われるので，アドレスはコード生成には使用されない

〔リスト 1〕MASM のストリング命令の記述例（つづき）

```
00000018  A6                    cmpsb
00000019  A6                    cmps    byte ptr [esi],byte ptr es:[edi]
0000001A  66| A7                cmpsw
0000001C  66| A7                cmps    word ptr [esi],word ptr es:[edi]
0000001E  A7                    cmpsd
0000001F  A7                    cmps    dword ptr [esi],dword ptr es:[edi]

00000020  AE                    scasb
00000021  AE                    scas    byte ptr es:[edi]
00000022  66| AF                scasw
00000024  66| AF                scas    word ptr es:[edi]
00000026  AF                    scasd
00000027  AF                    scas    dword ptr es:[edi]

00000028  AC                    lodsb
00000029  AC                    lods    byte ptr [esi]
0000002A  66| AD                lodsw
0000002C  66| AD                lods    word ptr [esi]
0000002E  AD                    lodsd
0000002F  AD                    lods    dword ptr [esi]

00000030  AA                    stosb
00000031  AA                    stos    byte ptr es:[edi]
00000032  66| AB                stosw
00000034  66| AB                stos    word ptr es:[edi]
00000036  AB                    stosd
00000037  AB                    stos    dword ptr es:[edi]

00000038  6C                    insb
00000039  6C                    ins     byte ptr es:[edi],dx
0000003A  66| 6D                insw
0000003C  66| 6D                ins     word ptr es:[edi],dx
0000003E  6D                    insd
0000003F  6D                    ins     dword ptr es:[edi],dx

00000040  6E                    outsb
00000041  6E                    outs    dx,byte ptr [esi]
00000042  66| 6F                outsw
00000044  66| 6F                outs    dx,word ptr [esi]
00000046  6F                    outsd
00000047  6F                    outs    dx,dword ptr [esi]

00000048  F3/ A4                rep   movsb
0000004A  F3/ A4                rep   movs      byte ptr es:[edi],byte ptr [esi]
0000004C  F3/ 66| A5            rep   movsw
0000004F  F3/ 66| A5            rep   movs      word ptr es:[edi],word ptr [esi]
00000052  F3/ A5                rep   movsd
00000054  F3/ A5                rep   movs      dword ptr es:[edi],dword ptr [esi]

00000056  F3/ A6                repe  cmpsb
00000058  F3/ A6                repe  cmps      byte ptr [esi],byte ptr es:[edi]
0000005A  F3/ 66| A7            repe  cmpsw
0000005D  F3/ 66| A7            repe  cmps      word ptr [esi],word ptr es:[edi]
00000060  F3/ A7                repe  cmpsd
00000062  F3/ A7                repe  cmps      dword ptr [esi],dword ptr es:[edi]

00000064  F3/ AE                repz  scasb
00000066  F3/ AE                repz  scas      byte ptr es:[edi]
00000068  F3/ 66| AF            repz  scasw
0000006B  F3/ 66| AF            repz  scas      word ptr es:[edi]
0000006E  F3/ AF                repz  scasd
00000070  F3/ AF                repz  scas      dword ptr es:[edi]

00000072  F2/ A6                repne cmpsb
00000074  F2/ A6                repne cmps      byte ptr [esi],byte ptr es:[edi]
00000076  F2/ 66| A7            repne cmpsw
00000079  F2/ 66| A7            repne cmps      word ptr [esi],word ptr es:[edi]
0000007C  F2/ A7                repne cmpsd
0000007E  F2/ A7                repne cmps      dword ptr [esi],dword ptr es:[edi]

00000080  F2/ AE                repnz scasb
00000082  F2/ AE                repnz scas      byte ptr es:[edi]
00000084  F2/ 66| AF            repnz scasw
00000087  F2/ 66| AF            repnz scas      word ptr es:[edi]
0000008A  F2/ AF                repnz scasd
0000008C  F2/ AF                repnz scas      dword ptr es:[edi]

                            end
```

CMPS 命令は，他の命令とは異なり，オペランドが「SOU，DEST」の順になっているので注意

REP，REPcc 命令を使用する場合は，MOVS などの動作命令の前に記述する

〔リスト2〕gasのストリング命令の記述例

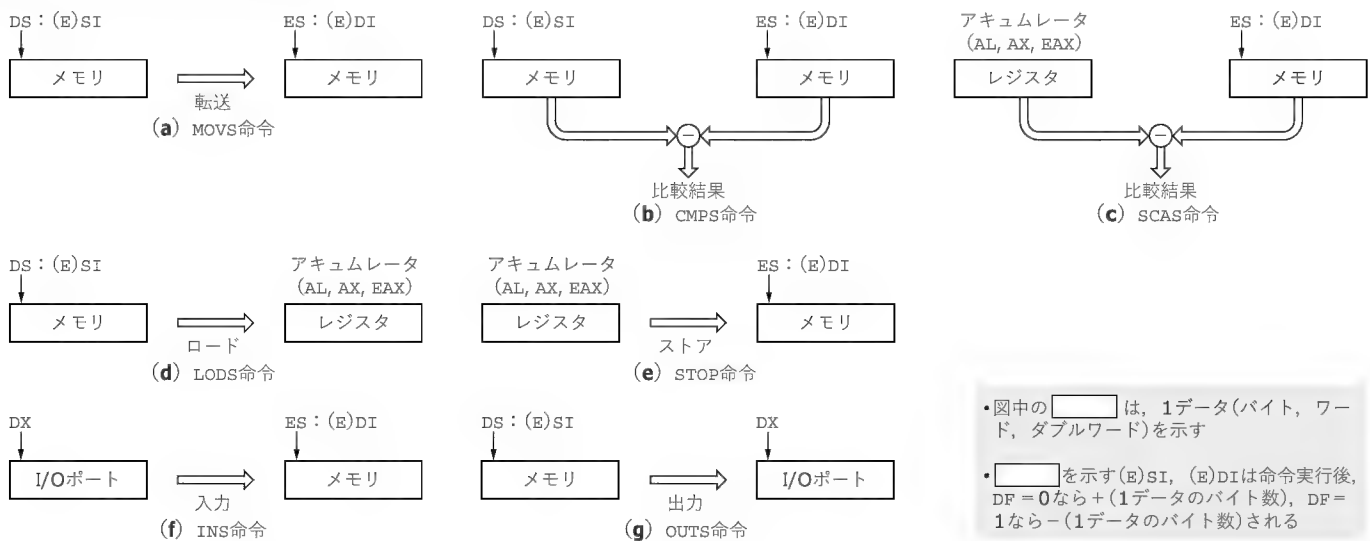
```
 1                    .text
 2 0000 A4                    movsb
 3 0001 A4                    movsb   (%esi),%es:(%edi)
 4 0002 66A5                  movsw
 5 0004 66A5                  movsw   (%esi),%es:(%edi)
 6 0006 A5                    movsl
 7 0007 A5                    movsl   (%esi),%es:(%edi)
 8
 9 0008 A6                    cmpsb
10 0009 A6                    cmpsb   %es:(%edi),(%esi)
11 000a 66A7                  cmpsw
12 000c 66A7                  cmpsw   %es:(%edi),(%esi)
13 000e A7                    cmpsl
14 000f A7                    cmpsl   %es:(%edi),(%esi)
15
16 0010 AE                    scasb
17 0011 AE                    scasb   %es:(%edi)
18 0012 66AF                  scasw
19 0014 66AF                  scasw   %es:(%edi)
20 0016 AF                    scasl
21 0017 AF                    scasl   %es:(%edi)
22
23 0018 AC                    lodsb
24 0019 AC                    lodsb   (%esi)
25 001a 66AD                  lodsw
26 001c 66AD                  lodsw   (%esi)
27 001e AD                    lodsl
28 001f AD                    lodsl   (%esi)
29
30 0020 AA                    stosb
31 0021 AA                    stosb   %es:(%edi)
32 0022 66AB                  stosw
33 0024 66AB                  stosw   %es:(%edi)
34 0026 AB                    stosl
35 0027 AB                    stosl   %es:(%edi)
36
37 0028 6C                    insb
38 0029 6C                    insb    %dx,%es:(%edi)
39 002a 666D                  insw
40 002c 666D                  insw    %dx,%es:(%edi)
41 002e 6D                    insl
42 002f 6D                    insl    %dx,%es:(%edi)
43
44 0030 6E                    outsb
45 0031 6E                    outsb   (%esi),%dx
46 0032 666F                  outsw
47 0034 666F                  outsw   (%esi),%dx
48 0036 6F                    outsl
49 0037 6F                    outsl   (%esi),%dx
50
51 0038 F3A4                  rep    movsb
52 003a F3A4                  rep    movsb     (%esi),%es:(%edi)
53 003c F366A5                rep    movsw
54 003f F366A5                rep    movsw     (%esi),%es:(%edi)
55 0042 F3A5                  rep    movsl
56 0044 F3A5                  rep    movsl     (%esi),%es:(%edi)
57
58 0046 F3A6                  repe   cmpsb
59 0048 F3A6                  repe   cmpsb     %es:(%edi),(%esi)
60 004a F366A7                repe   cmpsw
61 004d F366A7                repe   cmpsw     %es:(%edi),(%esi)
62 0050 F3A7                  repe   cmpsl
63 0052 F3A7                  repe   cmpsl     %es:(%edi),(%esi)
64
65 0054 F3AE                  repz   scasb
66 0056 F3AE                  repz   scasb     %es:(%edi)
67 0058 F366AF                repz   scasw
68 005b F366AF                repz   scasw     %es:(%edi)
69 005e F3AF                  repz   scasl
70 0060 F3AF                  repz   scasl     %es:(%edi)
71
72 0062 F2A6                  repne  cmpsb
73 0064 F2A6                  repne  cmpsb     %es:(%edi),(%esi)
74 0066 F266A7                repne  cmpsw
75 0069 F266A7                repne  cmpsw     %es:(%edi),(%esi)
76 006c F2A7                  repne  cmpsl
77 006e F2A7                  repne  cmpsl     %es:(%edi),(%esi)
78
79 0070 F2AE                  repnz  scasb
80 0072 F2AE                  repnz  scasb     %es:(%edi)
81 0074 F266AF                repnz  scasw
82 0077 F266AF                repnz  scasw     %es:(%edi)
83 007a F2AF                  repnz  scasl
84 007c F2AF                  repnz  scasl     %es:(%edi)
```

gasでは，b(バイト)，w(ワード)，l(ロング)で1データのサイズを指定する

必要ならオペランドを記述することもできるが，その場合は，
DESTは　%es:(%edi)
SOUは　(%esi)
のみが使用可能となっている

gasでもCMPS命令は，他の命令とは異なり，オペランドが「DEST，SOU」の順になっているので注意

〔図2〕ストリング命令の動作

(a) MOVS命令　(b) CMPS命令　(c) SCAS命令　(d) LODS命令　(e) STOP命令　(f) INS命令　(g) OUTS命令

ステータスフラグは減算結果にしたがい設定されます．

● SCAS命令

アキュムレータ(AL，AX，EAX)とDESTで指定されたバイトストリング1データ(1バイト，1ワード，1ダブルワード)の比較を行います〔**図2(c)**〕．比較は，アキュムレーター DESTの減算で行い，減算の結果である差は捨てられます．そのため，アキュムレータもDESTも変化しません．ステータスフラグは減算結果にしたがい設定されます．

● LODS命令

アキュムレータ(AL，AX，EAX)にSOUで指定されたバイトストリング1データ(1バイト，1ワード，1ダブルワード)をロードします〔**図2(d)**〕．

● STOS命令

アキュムレータ(AL，AX，EAX)の値をDESTで指定されたバイトストリング1データ(1バイト，1ワード，1ダブルワード)にストアします〔**図2(e)**〕．この命令は，繰り返し指定のREP命令を使用することで，メモリ上の指定領域を同じ値で高速に埋めることができます．

● INS命令

レジスタDXが示すI/Oポートからデータ(DESTと同じサイズ)を入力し，DESTで指定されたバイトストリング1データ(1バイト，1ワード，1ダブルワード)にストアします〔**図2(f)**〕．レジスタDXはINS命令実行中，変化しません．

● OUTS命令

レジスタDXが示すI/Oポートへ，SOUで指定されたバイトストリング1データ(1バイト，1ワード，1ダブルワード)を出力します〔**図2(g)**〕．レジスタDXはOUTS命令実行中，変化しません．

● REP命令

レジスタ(E)CXで指定された回数，後にある動作命令を実行します．つまり，後にある動作命令を1回実行するとレジスタ(E)CXは−1され，それがゼロになるまで繰り返されます．16ビットプログラムの場合レジスタCXが，32ビットプログラムの場合レジスタECXが使用されます．

動作命令としてはMOVS，LODS，STOS，INS，OUTSが指定できます(**図3**)．たとえば，Windowsの32ビットのMASMのプログラムで，レジスタESIで転送元の先頭アドレス，レジスタEDIで転送先の先頭アドレスを指定したとして，これを100ワード分，転送するのなら，

```
MOV  ECX, 100
CLD
REP  MOVSW
```

となります．

このREP命令をLODS命令，INS命令，OUTS命令で使用する場合，次のような問題があります．

(1) LODS命令でのREP命令の使用

LODS命令は，転送先がアキュムレータ(AL，AX，EAX)に固定されているため，この命令で繰り返し指定を使用しても意味がありません．

(2) INS命令でのREP命令の使用

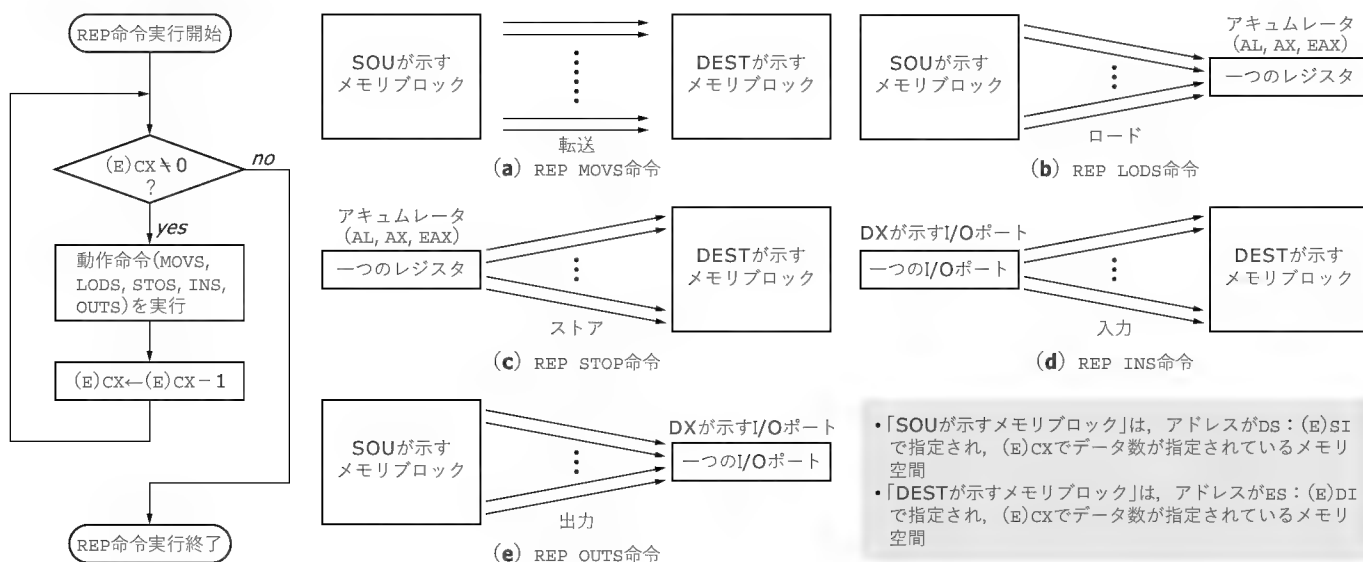
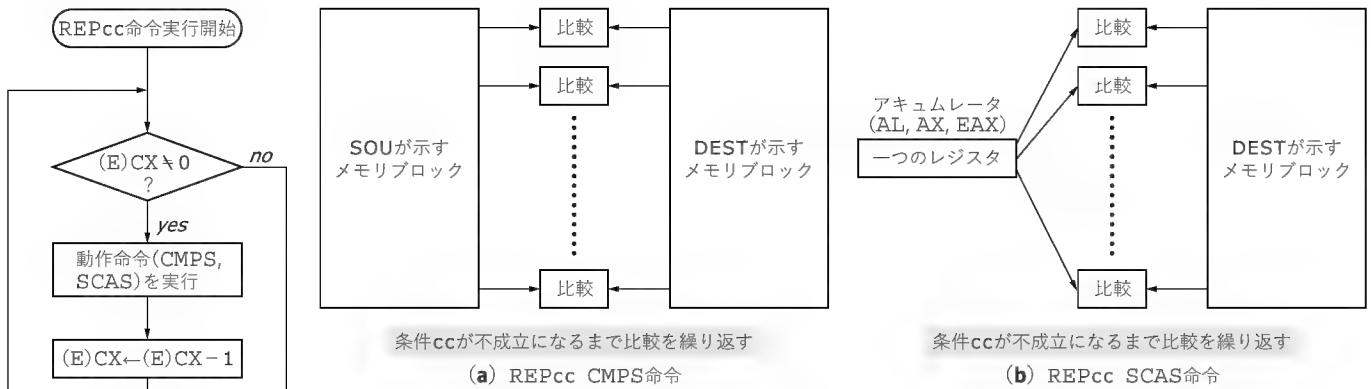
指定されたポートがハードウェア的にI/Oリードで同期入力するようになっていないと，INS命令でREP命令を使用してもむだの多い無意味なI/O入力となってしまいます．

(3) OUTS命令でのREP命令の使用

指定されたポートがハードウェア的にI/Oライトで同期出力するようになっていないと，OUTS命令でREP命令を使用しても周辺I/Oにとって無意味なI/O出力となってしまいます．それればかりか，周辺I/Oによっては高速な連続したI/Oライトは，ハードウェア的な破壊を起こす場合もあるので，OUTS命令でのREP命令の使用には十分注意する必要があります．

〔図3〕REP命令の動作

〔図4〕REPcc命令の動作

(a) REPcc CMPS命令　(b) REPcc SCAS命令

● REPcc命令

REP命令の動作に，ccで指定される繰り返し条件が付いたものです．ccにはZ，E，NZ，NZが入ります．Z，E，NZ，NZはJcc命令の条件と同じです．つまり，REPZ，REPEは(E)CX＝0かZF＝0となるまで動作命令を繰り返し実行します．また，REPNZ，REPNEは(E)CX＝0かZF＝1となるまで動作命令を繰り返し実行します(**図4**)．

動作命令としては，その性格上CMPS，SCAS命令のみがこのREPcc命令を使用することができます．

システム命令の概要

システム命令には，**表2**のような命令があります．システム命令は，その名のようにCPU自体をコントロールする命令です．つまり，システム命令はOSレベルで使用される命令で，WindowsやLinux上で通常実行されるアプリケーションプログラムでは，システム命令は一切使用しませんし，実行もできません．そのためこのシステム命令を，OS上で実行されるアプリケーションレベルのプログラムでチョット試すといったこともできないわけです．

ですから，この連載のターゲットであるWindowsやLinux上で実行されるアプリケーションプログラムの作成レベルでは，このシステム命令は使用されることがないので，ここでは**表2**で示したシステム命令の内，おもな命令について，その概要を説明することにします．

● メモリマネージメントレジスタのロード/ストア

表2のLIDT，SIDT，LGDT，SGDT，LLDT，SLDT，LTR，STRの8命令は，メモリマネージメントレジスタのIDTR，

〔表2〕x86系の32ビットCPUのシステム命令の一覧

インストラクション名	動　作
LIDT	割り込みディスクリプタテーブル(IDT)レジスタのロード
SIDT	割り込みディスクリプタテーブル(IDT)レジスタのストア
LGDT	グローバルディスクリプタテーブル(GDT)レジスタのロード
SGDT	グローバルディスクリプタテーブル(GDT)レジスタのストア
LLDT	ローカルディスクリプタテーブル(LDT)レジスタのロード
SLDT	ローカルディスクリプタテーブル(LDT)レジスタのストア
LTR	タスクレジスタのロード
STR	タスクレジスタのストア
MOV	オペランドにCR0～CR4が指定されている場合は，制御レジスタのロードとストア
LMSW	マシンステータスワード(MSW：CR0の下位16ビット)のロード
SMSW	マシンステータスワード(MSW：CR0の下位16ビット)のストア
CLTS	タスクスイッチフラグのクリア
ARPL	特権レベル要求の調整
LAR	アクセス権バイトのロード
LSL	セグメントリミットのロード
VERR	セグメントが読み出し可能か調べる
VERW	セグメントが書き込み可能か調べる
MOV	オペランドにDR0～DR7が指定されている場合は，デバッグレジスタのロードとストア
INVD	キャッシュを無効にする(ライトバックなし)
WBINVD	キャッシュを無効にする(ライトバックあり)
INVLPG	TLBエントリを無効にする
LOCK	プリフィックスとして指定する．LOCKの後にある命令の実行中は，LOCK#信号を出しバスをロックする
HLT	CPUを停止する
RSM	システムマネジメントモード(SSM)の割り込みからの復帰
RDMSR	MSRレジスタのリード
WRMSR	MSRレジスタのライト
RDPMC	性能モニタリングカウンタのリード
RDTSC	タイムスタンプカウンタのリード
SYSENTER	高速システムコールの入口
SYSEXIT	高速システムコールの出口

GDTR，LDTR，TRに対するロード/ストアを行う命令です．

(1) LIDT，SIDT命令

LIDT命令は，レジスタIDTRにメモリ上にあるIDTのベースアドレスとリミットをロードする命令で，SIDT命令は，その逆のレジスタIDTRに格納されているIDTのベースアドレスとリミットをメモリにストアする命令です(**図5**)．

IDT(Interrupt Descriptor Table)は，メモリ上にあるゲートディスクリプタと呼ばれる割り込みや例外が発生したときの飛び先を示すディスクリプタを格納しているテーブルのことです．

(2) LGDT，SGDT命令

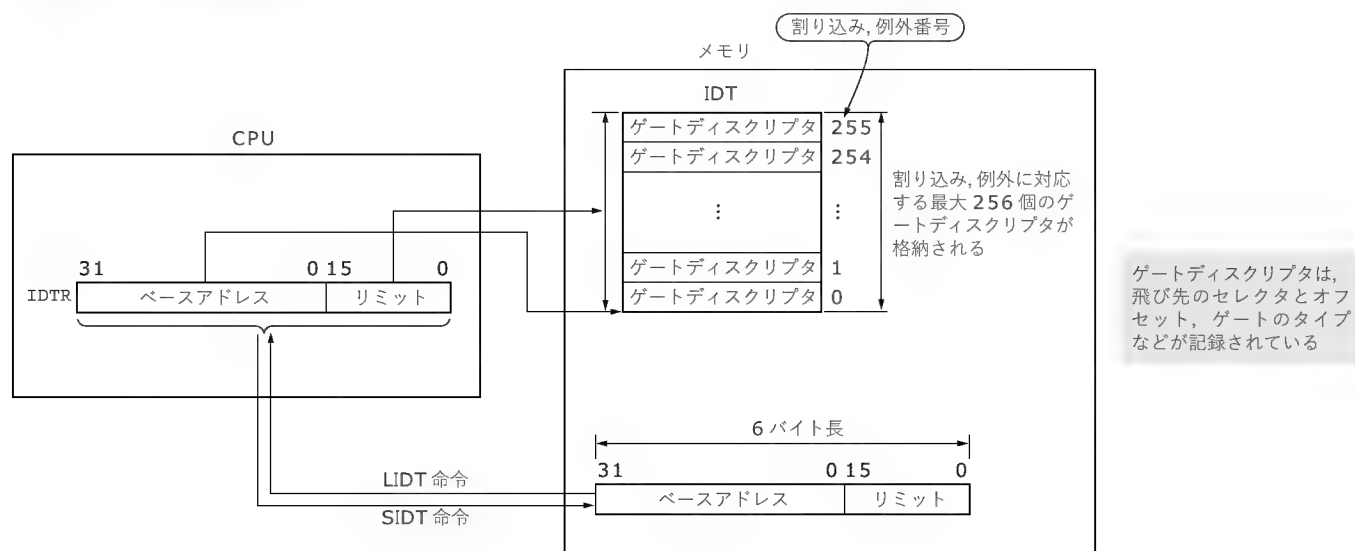
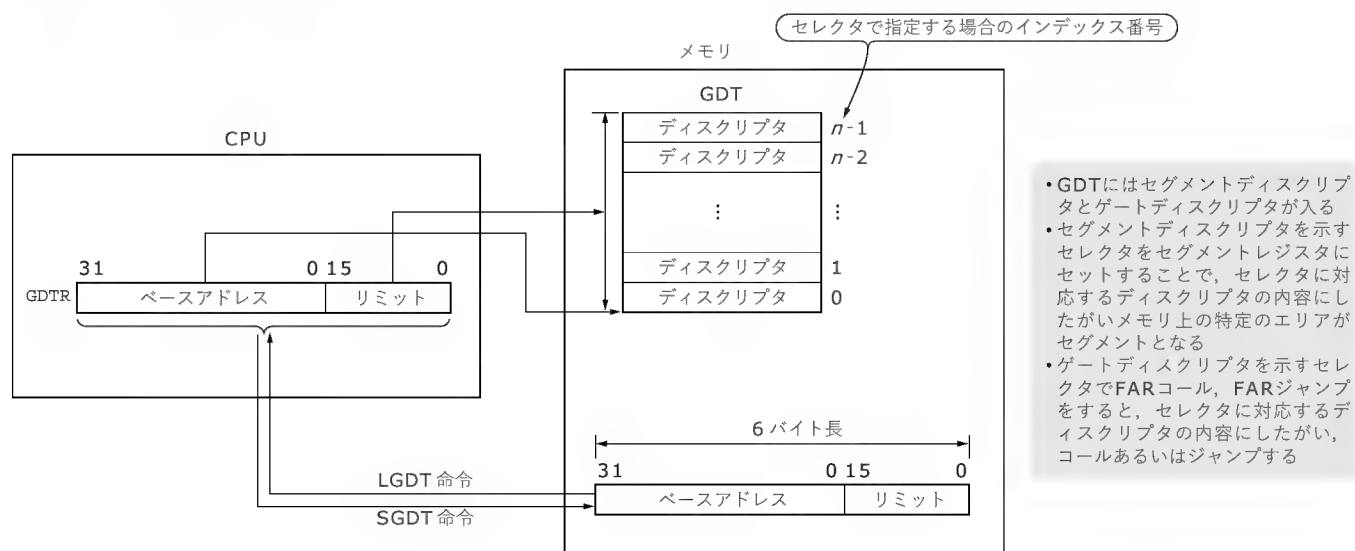
LGDT命令は，レジスタGDTRにメモリ上にあるGDTのベースアドレスとリミットをロードする命令で，SGDT命令は，その逆のレジスタGDTRに格納されているGDTのベースアドレスとリミットをメモリにストアする命令です(**図6**)．

GDT(Global Descriptor Table)は，メモリ上にあるセグメントディスクリプタと呼ばれるセグメントを管理するディスクリプタやゲートディスクリプタと呼ばれるCALLやJMPの飛び先を示すディスクリプタを格納しているテーブルのことです．

GDTは，その名のように個々のタスクで共通に使用されるグローバルなディスクリプタを格納するためのテーブルです．

(3) LLDT，SLDT命令

LLDT命令は，レジスタLDTRにGDT上にあるLDTディスクリプタを示すセレクタをロードする命令で，SLDT命令は，その逆のレジスタLDTRに格納されているLDTディスクリプタを示すセレクタをメモリにストアする命令です(**図7**)．

〔**図5**〕**LIDT，SIDT命令の動作**

〔**図6**〕**LGDT，SGDT命令の動作**

LDT (Local Descriptor Table) は，メモリ上にあるセグメントディスクリプタやゲートディスクリプタを格納しているテーブルのことです．LDT は，その名のように個々のタスクでのみ使用されるローカルなディスクリプタを格納するためのテーブルです．

ここで一つ注意事項があります．レジスタ IDTR やレジスタ GDTR では，直接テーブルのベースアドレスとリミットをロード/ストアしていました．しかしレジスタ LDTR は，GDT 上にある LDT ディスクリプタを示すセレクタをロード/ストアしています．そのため，LDT 自体のベースアドレスとリミットは，レジスタ LDTR 上のセレクタが示す LDT ディスクリプタにあるベースアドレスとリミットから得ることになります．

(4) LTR，STR 命令

LTR 命令は，レジスタ TR に GDT 上にある TSS ディスクリプタを示すセレクタをロードする命令で，STR 命令は，その逆のレジスタ TR に格納されている TSS ディスクリプタを示すセレクタをメモリにストアする命令です(**図 8**，次頁)．

レジスタ TR は，現在実行中のタスクを示すもので，実行中のタスクの TSS ディスクリプタを示すセレクタが格納されています．

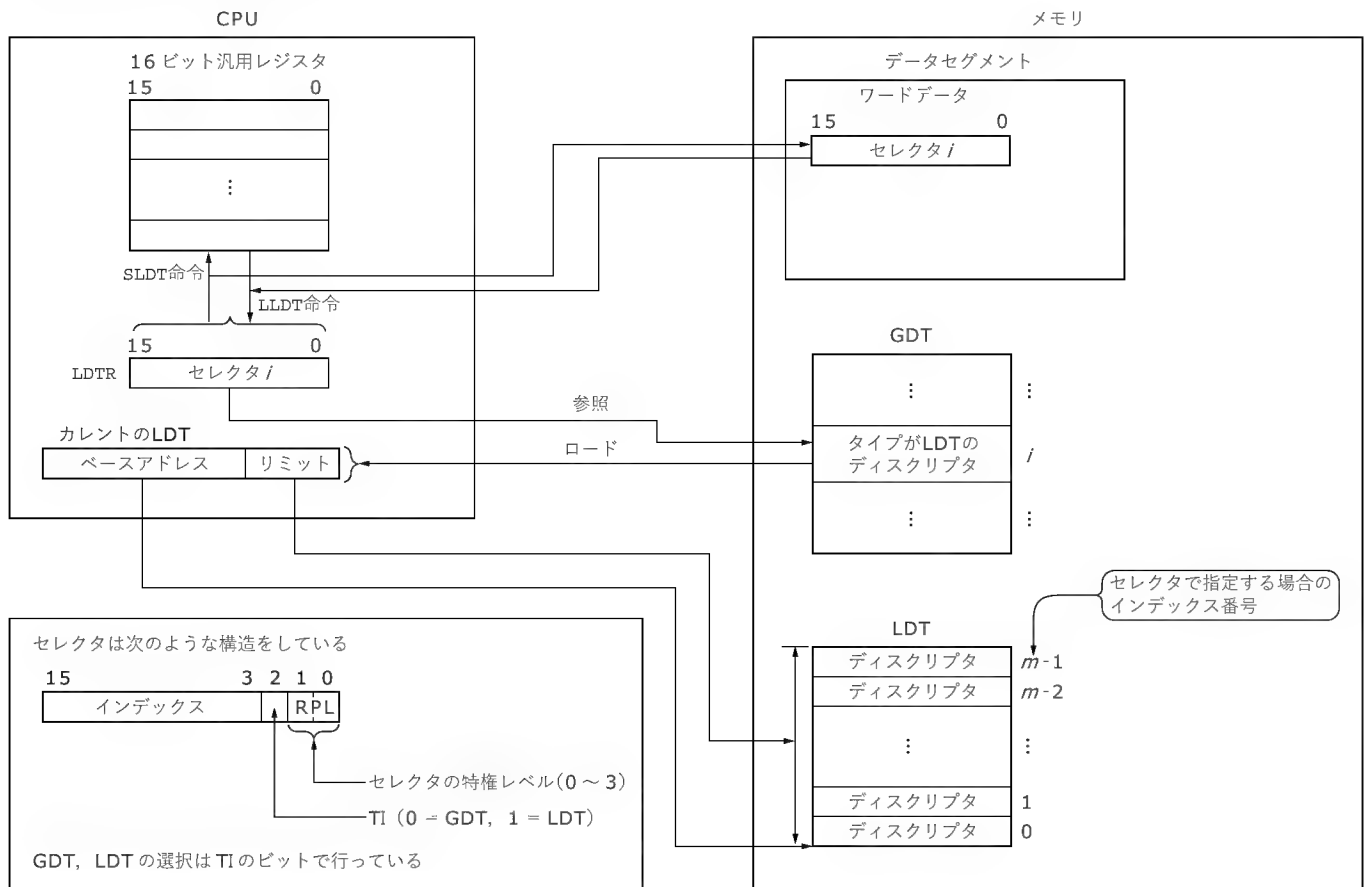
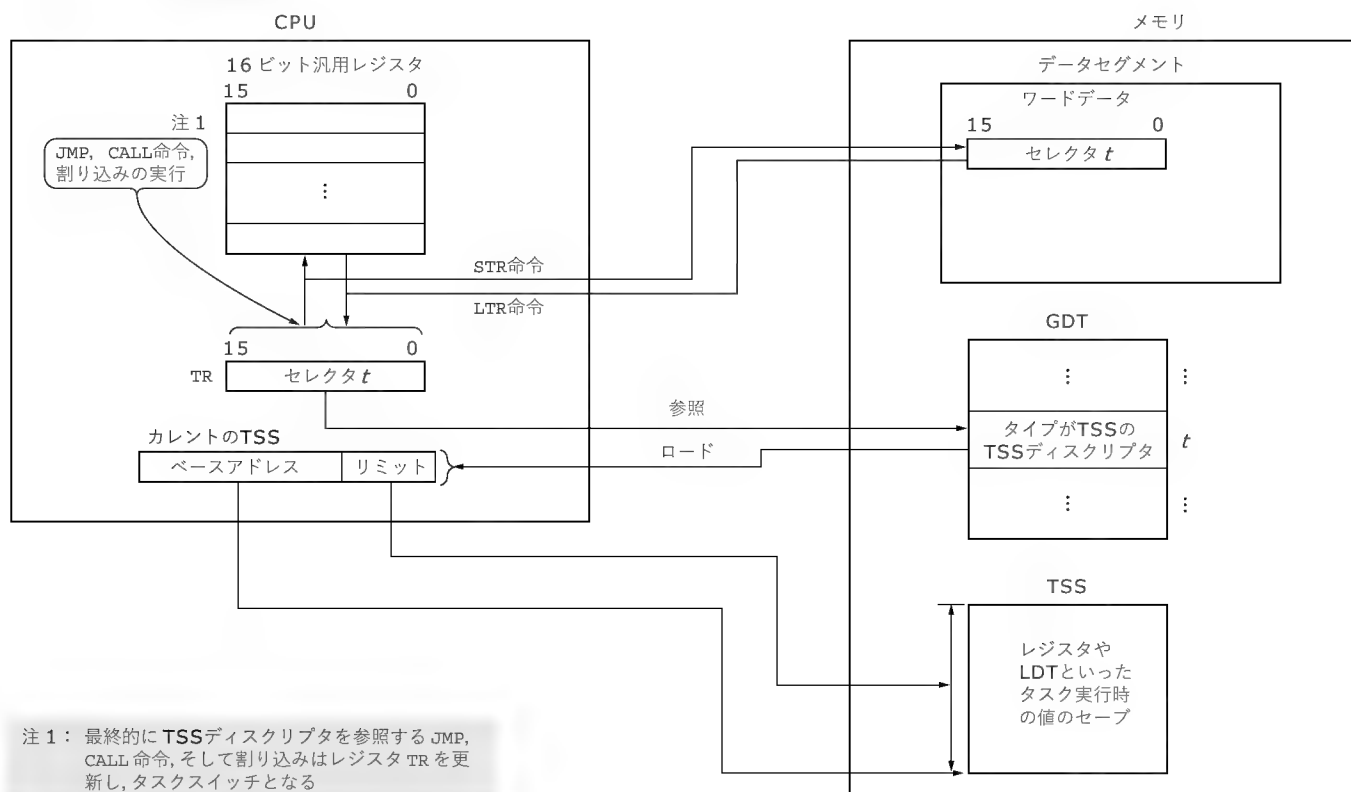
TSS (Task State Segment) は，タスクを管理するためのセグメントで，個々のタスクがこの TSS を持っています．TSS にはタスク実行時のレジスタ値や LDT といった情報がセーブされています．通常，タスクスイッチは，TSS ディスクリプタを示すセレクタを使った CALL や JMP，割り込みで行います．この LTR，STR 命令は，初期タスクの設定や現在のタスクを取得するといった用途に使用されます．

● 制御レジスタに対するロード/ストア

CPU の制御レジスタ (CR0，CR2 ～ 4) に対するロード/ストアは，MOV 命令により行います．MOV 命令でオペランドに CR0，CR2 ～ 4 を指定することでアクセスできます．ただし，制御レジスタに対する MOV 命令では，もう一方のオペランドは 32 ビットの汎用レジスタのみ指定可能となっています．

制御レジスタ CR0 の下位 16 ビットに対してのロード/ストアには，専用の命令 LMSW，SMSW があります．LMSW 命令がロード，SMSW 命令がストアとなります．この LMSW，SMSW の二つの命令は 16 ビット CPU の 80286 のときに使われていたもので，386 以降の 32 ビット CPU にも互換性のために残されています．

そのため，386 以降の 32 ビット CPU では LMSW，SMSW 命令の代わりに，オペランドに CR0 を指定した MOV 命令を使用す

〔**図 7**〕**LLDT，SLDT 命令の動作**

〔図8〕LTR，STR命令の動作

ることが推奨されています．

● デバッグレジスタに対するロード/ストア

CPUのデバッグレジスタ（DR0〜7）に対するロード/ストアも，MOV命令により行います．MOV命令でオペランドにDR0〜7を指定することでアクセスできます．ただし，デバッグレジスタに対するMOV命令でも，もう一方のオペランドは32ビットの汎用レジスタのみ指定可能となっています．

● ディスクリプタ上の情報取得

セグメントディスクリプタ上には，セグメントを示すベースアドレスやリミット，セグメントの特性を示したアクセス権バイトといった情報が格納されています．このセグメントディスクリプタ上のアクセス権バイトの取得にはLAR命令，リミットの取得にはLSL命令を使用することができます．

● セグメントが読み出し/書き込み可能か調べる

セレクタが示すセグメントが読み出し/書き込み可能かを調べることができます．VERR命令を使用することでセグメントが読み出し可能かを調べることができ，VERW命令でセグメントが書き込み可能かを調べることができます．

● セレクタの特権レベルの調整

ARPL命令を使用することで，セレクタ内に記憶されている特権レベルを調整することができます．このとき，ARPL命令はオペランドとして指定された二つのセレクタの一方を基準とし，もう一方のセレクタの特権レベルを調整します．

● CR0のTSフラグのクリア

CLTS命令を実行すると制御レジスタCR0にあるTSフラグがクリアされます．TSフラグはタスクスイッチが発生するごとにセットされ，一度セットされると自動的にクリアされることはありません．

このTSフラグは，システムプログラムがタスクスイッチが発生したか否かを知るのに使用し，タスクスイッチが発生したことをシステムプログラムが感知した後は，CLTS命令でTSフラグをクリアします．

● キャッシュの制御

INVD命令は，内部キャッシュをフラッシュし，外部キャッシュに対してフラッシュを要求します．このINVD命令を実行すると，内部キャッシュおよび外部キャッシュ内にあるデータはすべて捨てられてしまいます．

WBINVD命令も，内部キャッシュをフラッシュし，外部キャッシュに対してフラッシュを要求します．ただし，このWBINVD命令は内部キャッシュのフラッシュに先立ち，内部キャッシュ内にあるデータをメインメモリにライトバックします．

また，外部キャッシュに対してもフラッシュ要求に先立ち，外部キャッシュ内にあるデータのメインメモリへのライトバックを要求します．

● CPUを停止させる

HLT命令を実行すると，CPUは停止（HALT）状態になりま

す．停止状態は，許可されている割り込みやNMI，リセットにより解除され，実行を再開します．ただし，割り込みやNMIで実行を再開した場合は，HLT命令の次の命令から実行再開されます．

● MASMやgasでのシステム命令の記述

はじめにも述べたように，WindowsやLinux上で通常実行されるアプリケーションプログラムでは，このシステム命令はいっさい使用しません．しかし，MASMやgasでシステム命令がアセンブルできるかどうかは別の話です．この連載で使用しているMASMやgasでもすべてではありませんが，**表2**(p.127)で示したおもなシステム命令はアセンブルすることができます．

MASMの場合は，アセンブル対象のCPUを指定するディレクティブ命令で，最後に‘p’を付けることでシステム命令がアセンブルできます．たとえば，

```
.586p
```

のように指定します．

gasは，そのままの状態でシステム命令のアセンブルが可能です．ただし，アセンブルできるのと実行できるのとでは別の問題なので，WindowsやLinux上でシステム命令をアセンブルしても，実行することはできません．

● システム命令の詳細を知るには？

OSを使用しないROM化するようなプログラムの作成では，CPU自体の制御も，ユーザー作成のプログラムがする必要があります．そのようなとき，このシステム命令を使用する必要がある場合もあります．

このシステム命令を使うことで，x86系CPUがもつプロテクトモードやタスクの制御，そして仮想記憶やメモリキャッシュ制御の制御といったことが行えるようになります．しかし，システム命令を本当の意味の使いこなすためには，x86系CPU自体の詳細な動作を理解，あるいは勉強する必要があるといえます．

このx86系CPUがもつプロテクトモードやタスク，仮想記憶やメモリキャッシュといったことやシステム命令の動作の詳細は，インテルが発行しているCPUのマニュアルにその記述があります．

CPUのマニュアルは，PDF形式のファイルでよければ2003年5月末現在，インテルのホームページから入手することができます．まず，`http://www.intel.co.jp/`でインテルのホームページに行き，「デベロッパ」，「ハードウェア設計」，「プロセッサ」と進み，必要なプロセッサを選択，左下にある「マニュアル(日本語/英語)」で必要なマニュアルをダウンロードすることができます．

このマニュアルには，システム命令のほかに，いままでこの連載で説明してきた汎用命令やこれから解説する予定のFPU命令やSIMD命令なども詳細に解説されているので，興味のある方は，ぜひご利用ください．

*　　*

次回は，x86系CPUがもつ浮動小数点演算に関するFPU命令について説明する予定です．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

Pentium4/Intel Xeonにおける性能モニタ機能を利用した

メモリプロファイリングツールを開発する ―― 基礎知識編

吉岡弘隆

はじめに

最近のマイクロプロセッサは，性能モニタ用のハードウェアを内蔵している．また，マイクロプロセッサがますます高度化，複雑化していけばいくほど，アプリケーションから見たマイクロプロセッサの動作の理解が難しくなってきている．

初期のマイクロプロセッサと異なり最近のプロセッサは，①深いパイプラインをもつ，②投機的な実行をする，③スーパスカラで同時に複数命令を実行する，④アウトオブオーダ実行をする，などの特徴をもつ．さらに，キャッシュをはじめとするメモリ階層やマルチプロセッサなど，より高度な機能が実装されている．

これらの機能を十分に活用しつつ性能を向上させるために，ハードウェア性能モニタ機能が重要になってきている．本稿では，解説を2回に分け，まず今回は，Intelの32ビットマイクロプロセッサ(IA-32と記す)のハードウェア性能モニタ機能について紹介し，次にIntel Pentium4/Intel Xeonプロセッサで強化された機能について詳細に解説する．次回は，Pentium 4/Intel Xeonの性能モニタ機能を利用してLinux上に実装したメモリプロファイリングツールについて，実装，利用方法などを解説する．

1．IA-32における性能モニタ機能

Intelの32ビットマイクロプロセッサは，モデル固有のハードウェア性能モニタ機能をもつ．まず，IA-32の性能モニタ機能を紹介する．

〔リスト1〕**RDTSC命令の使用例**(gccでのコーディング)

```
#define rdtscll(val) ¥
     __asm__ __volatile__("rdtsc" : "=A" (val))

unsigned long long before;
unsigned long long after;

rdtscll(before);
/* 計測する部分 */
rdtscll(after);

diff=after-before; /* 実行クロック数 */
```

1.1　各種性能カウンタ

性能カウンタ(PMC)はPentiumから実装され，さまざまなハードウェアイベントの計測を可能にしている．計測できるイベントはアーキテクチャモデルによって異なる．最新のPentium 4およびIntel Xeonプロセッサでは，40ビットの性能カウンタを18個もち，多くのイベントを同時に計測することができるようになった(計測できるイベント例：分岐命令数，分岐予測失敗数，バストランザクション数，キャッシュミス数，TLBミス数，実行命令数など多数)．

タイムスタンプカウンタ(TSC)は，ハードウェアリセット時に0から開始し，プロセッサのクロックサイクルごとに増加する64ビットのレジスタである．RDTSC命令によって，カウンタの値を読む(非特権命令)．ユーザーモードからも読めるので，簡単に実行時のクロック数を計測するのに利用できる．たとえば，あるルーチンの実行コストは入口と出口でTSCを読み，その差が実行コスト(サイクル数)になる(**リスト1**)．タイムスタンプカウンタを読む方法は，簡単にしかも正確に実行コストを得られる．Pentium IIIで計測できるイベント例を**表1**に示す．

1.2　Pentium4における性能モニタリング機能

性能カウンタはPentiumから実装されたが，次に示すようにいくつかの限界があった．

- 同時に計測できるイベント数が少ない(Pentium IIIで二つ)

〔表1〕**Pentium IIIで計測できるイベント例**

コード	ニモニック
0x43	DATA_MEM_REFS：すべてのメモリアクセス数
0x48	DCU_MISS_OUTSTANDING：DCUミスサイクル数
0x80	IFU_IFETCH：命令フェッチ数
0x81	IFU_IFETCH_MISS：ミスした命令フェッチ数
0x85	ITLB_MISS_ITLB：ミス数
0x86	IFU_MEM_STALL：命令フェッチミスサイクル数
0x87	ILD_STALL：命令length decoderがストールしたサイクル数
0x29	L2_LD：L2データロード
0x2A	L2_ST：L2データストア
0xC0	INST_RETIRED：実行命令数
0xC2	UOPS_RETIRED：実行マイクロ命令数
0xA2	RESOURCE_STALLS：ストールサイクル数
0x79	CPU_CLK_UNHALTED：クロック数

●リタイアせずにキャンセルされた命令のイベントも計測する
●イベントサンプリングの精度(粒度)が荒い

Pentium III(P6アーキテクチャと呼ばれている)系プロセッサは投機的な実行を行う．したがって分岐予測が外れた場合，命令をキャンセルするが，キャンセルされた命令が引き起こしたイベントもカウントする．予測がはずれた側の命令が引き起こしたイベント(たとえばL2キャッシュミス)の回数も数えてしまう．この場合，当該イベントが多数発生するのは，分岐予測が外れたのがおもな原因なのか，それとも，L2キャッシュミスを多発させるようなプログラムなのかは，この情報だけでは判断できない．プログラムを改良するとき，分岐予測が外れないように改良すべきか，それともL2キャッシュミスを減らすような改良をすべきか，どちらにプライオリティを置くかなどが判断できないのである．

リタイア(コミット)した命令によって発生したイベントとキャンセルした命令によって発生したイベントを明確に分離できなければならない．

最近のプロセッサでは，イベントサンプリングをする場合，ある回数ごとに例外処理ルーチンが起動され，実際のプログラムカウンタ(PC)や各種レジスタの情報を収集する時点では，例外処理ルーチンのレイテンシおよび上記のプロセッサの特性により，取得したPCの値は実際のイベントを発生させたPCよりかなり先を行っている．参考文献2)によると，PentiumProで，取得したPCと実際のPCは25命令以上隔たって分布した．参考文献3)によれば，Pentium4では65命令以上実際の命令と隔たってサンプリングされた．プロセッサがより深いパイプラインをもつようになればなるほど，この隔たりは広くなると予想される．

このように，イベント発生の場所を正確に特定できないため，イベントが発生している時点の正確なコンテキストを入手できない．イベントが発生していることはわかっても，正確なコンテキストがわからないので，性能上何が問題かを特定することは，これらの情報だけでは困難である．イベントの発生の正確な特定と，その時点での精密なコンテキストの入手が求められていた．

上記の問題を解決するために，Pentium4/Intel Xeonプロセッサでは下記のような拡張を行っている．

●多くの性能カウンタ
●PEBS(Precise Event Based Sampling)

ここでは，Pentium4における性能モニタリングカウンタについて解説する．

性能モニタ機能はPentiumから導入されたが，Pentium/P6/Intel Xeon系それぞれ実装依存(model specific)で互換性はない．しかし，wrmsr/rdmsr/rdpmc命令によってイベントの選択，フィルタリング，計測，読み込みするという概念に違いはない．

Pentium4の性能モニタ機能は，各種イベントの検出器(event detectors)とカウンタからなる．性能モニタ機能を利用して各種ハードウェアイベントを計測する概要は，次のとおりである．

1) ESCR(Event Selection Control Registor)に計測するイベントとイベントマスクを設定する
2) 計測するモード(カーネルないしユーザー)をESCRのOS/USRフラグで設定する
3) 計測方法はCCCR(Counter Configutstion Control Registor)に設定するので，どのCCCRを選択するか，ESCRに設定する
4) CCCRのcompare/complement flagsおよびthresholdフィールドを設定する
5) 必要であれば，CCCRのedge flagを設定する
6) CCCRのEnable Flagを有効にする

ここでESCRが各種イベントの検出を行い，CCCRがカウンタの設定を行う．ESCRないしCCCRの設定はwrmsr命令で行い，6)の段階からパフォーマンスモニタはイベント計測を開始し，rdpmc命令によってパフォーマンスカウンタを読む．ESCRとCCCRのブロック図を**図1**に示す．

● 各種レジスタ

Pentium4とIntel Xeonプロセッサは，九つのペアの18個の性能カウンタをもつ．各ペアはイベントのサブセットとESCRに関連付けられている．カウンタのペアは下記の四つのグループに分類される．

1) BPUグループ
MSR_BPU_COUNTER0/MSR_BPU_COUNTER1/MSR_BPU_COUNTER2/MSR_BPU_COUNTER3
2) MSグループ
MSR_MS_COUNTER0/MSR_MS_COUNTER1/MSR_MS_COUNTER2/MSR_MS_COUNTER3
3) FLAMEグループ
MSR_FLAME_COUNTER0/MSR_FLAME_COUNTER1/MSR_FLAME_COUNTER2/MSR_FLAME_COUNTER3
4) IQグループ
MSR_IQ_COUNTER0/MSR_IQ_COUNTER1/MSR_IQ_COUNTER2/MSR_IQ_COUNTER3/MSR_IQ_COUNTER4/MSR_IQ_COUNTER5

各グループは，イベント検出器(ESCR)とカウンタ(およびCCCR)に関連付けられている．たとえば，BPU(Branch Prediction Unit)は，分岐予測に関連するイベントを検出し，カウントする．MSR_IQ_COUNTER4カウンタは，PEBS(Precise event-based sampling)のサポートに利用される．

各種レジスタの機能などを解説する．

1) ESCR(Event Selection Control Registors)
45個のMSR(Model Specific Registor)がある．モニタするイベントを選択する．
2) 18個のイベントをカウントするPerformance Counter MSR

〔図1〕ESCRとCCCRのブロック図(出典:`http://www.computer.org/micro/mi2002/pdf/m4072.pdf`)

BPU ESCR0 ESCR1 0
ISTEER ESCR0 ESCR1 1
IXLAT ESCR0 ESCR1 2
ITLB ESCR0 ESCR1 3
PMH ESCR0 ESCR1 4
MOB ESCR0 ESCR1 5
FSB ESCR0 ESCR1 6
BSU ESCR0 ESCR1 7
Counter block groups
BPU
CCCR/Conuter 0
CCCR/Conuter 1
CCCR/Conuter 2
CCCR/Conuter 3
MS ESCR0 ESCR1 0
TC ESCR0 ESCR1 1
TBPU ESCR0 ESCR1 2
MS
CCCR/Conuter 0
CCCR/Conuter 1
CCCR/Conuter 2
CCCR/Conuter 3
FLAME ESCR0 ESCR1 0
FIRM ESCR0 ESCR1 1
SAAT ESCR0 ESCR1 2
U2L ESCR0 ESCR1 3
DAC ESCR0 ESCR1 5
FLAME
CCCR/Conuter 0
CCCR/Conuter 1
CCCR/Conuter 2
CCCR/Conuter 3
IQ ESCR0 ESCR1 0
ALF ESCR0 ESCR1 1
RAT ESCR0 ESCR1 2
SSU ESCR0 3
CRU ESCR0 ESCR1 4
ESCR2 ESCR3 5
ESCR4 ESCR5 6
ESCR select values for CCCRs
IQ
CCCR/Conuter 0
CCCR/Conuter 1
CCCR/Conuter 2
CCCR/Conuter 3
CCCR/Conuter 4
CCCR/Conuter 5

がある.

3) 18個のPerformance Counter MSRに対応するCCCR MSRがある.各CCCRはカウント方法など,対応するperformance counter用に設定する.

IA-32 Intel Architecture Software Developer's Manual Volume 3:System Programming Guideの表15-4に,performance counter MSRと対応するCCCRおよびESCR MSRの表がある(`http://developer.intel.com/design/pentium4/manuals/245472.htm`を参照のこと).以降の図でキャプションの横にページ番号が入っているものがある場合,とくにことわりがなければ,上記マニュアルのページ番号を示している.

● **ESCR**

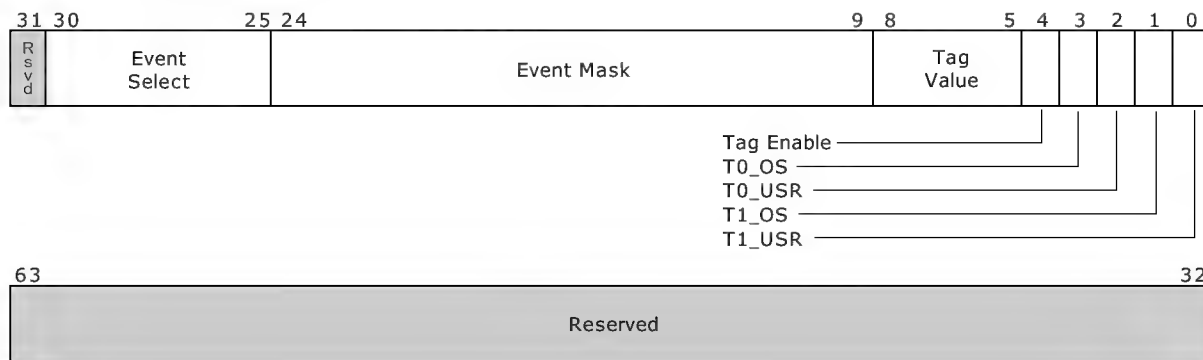
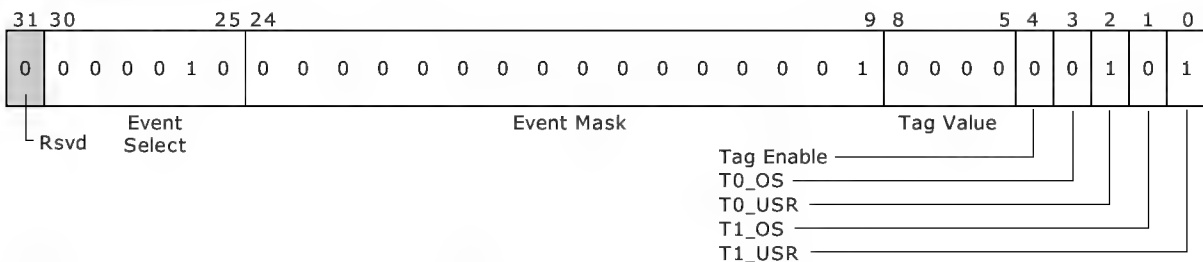
45個のESCR MSR(Model Specific Register)は計測するイベントを選択する.各ESCRは通常,性能カウンタのペアと関連付けられていて,各性能カウンタはいくつかのESCRと関連付けられている.

図2はESCR MSRのレイアウトを示している.ESCRはHyperThreadingサポートのため,ビット0/ビット1の利用が可能になっている.HyperThreadingをサポートしない,Pentium4などはビット0および1(T1_USRとT1_OS)は利用できないので注意が必要である.

Intel XeonではHyperThreadingがサポートされたため,一つの物理プロセッサに対し二つの論理プロセッサが存在し,それぞれを計測できるようになった.

図2で,T0_USRフラグ(ビット2)に1が設定されていると,論理プロセッサ0で実行しているcurrent privilege level(CPL)が1,2,3(ユーザーモード)のイベントがカウントされる.

T0_OSフラグ(ビット3)が設定されていると,論理プロセッサ0で実行しているCPLが0(カーネルモード)のイベントがカ

〔図2〕 **Event Selection Control Register (ESCR)** (p.15-55)

〔図3〕 **ESCRの設定例** (p.15-27)

```
0x04000204 == event select == 2 (instr_retired)
              event mask   == 1 Non bogus instruction, not tagged
              USR          == 5 user mode
```

上記例では，リタイヤした命令の〔参考文献1〕のTable A-2参照〕Non bogus instruction/not taggedでユーザーモードのイベントをカウントする．
対応するESCRはMSR_CRU_ESCR0ないしはMSR_CRU_ESCR1．
対応するカウンタは，

```
ESCR0 : 12, 13, 16
ESCR1 : 14, 15, 17
カウンタ              No.
MSR_IQ_COUNTER0: 12
MSR_IQ_COUNTER1: 13
MSR_IQ_COUNTER2: 14
MSR_IQ_COUNTER3: 15
MSR_IQ_COUNTER4: 16
MSR_IQ_COUNTER5: 17
```

ウントされる．

T1_USRフラグ(ビット0)が設定されていると，論理プロセッサ1で実行しているcurrent privilege level (CPL)が1，2，3(ユーザーモード)のイベントがカウントされる．

T1_OSフラグ(ビット1)が設定されていると，論理プロセッサ1で実行しているCPLが0(カーネルモード)のイベントがカウントされる．

両方(OSとUSR)設定されていると，両モードのイベントがカウントされる．

Tag Enable(ビット4)が設定されていると，at-retirementイベントのカウントを補助するuOPsのtaggingを可能にする．

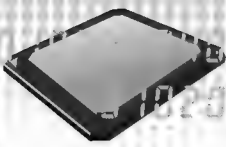
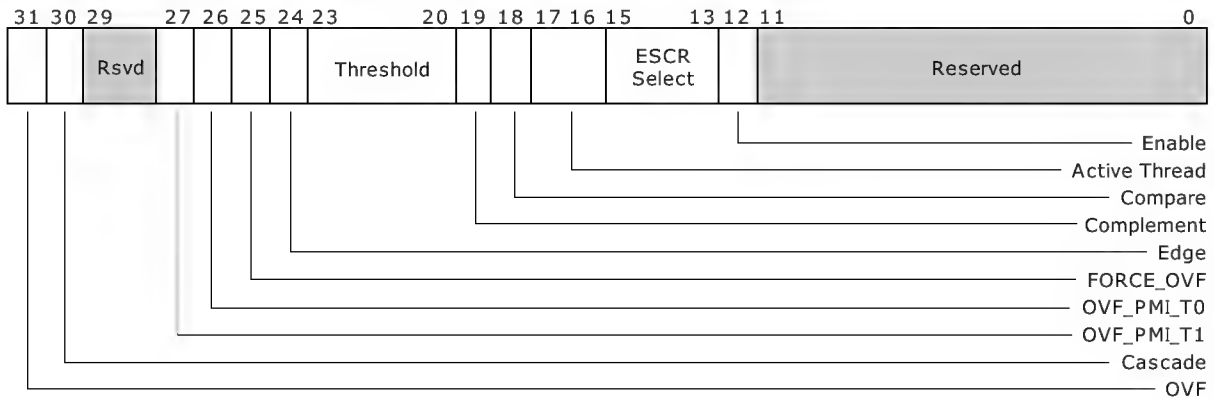
Tag Value field(ビット5～8)が設定されていると，at-retirementイベントをカウントするのに対応するuOPsのtag値を選択する．

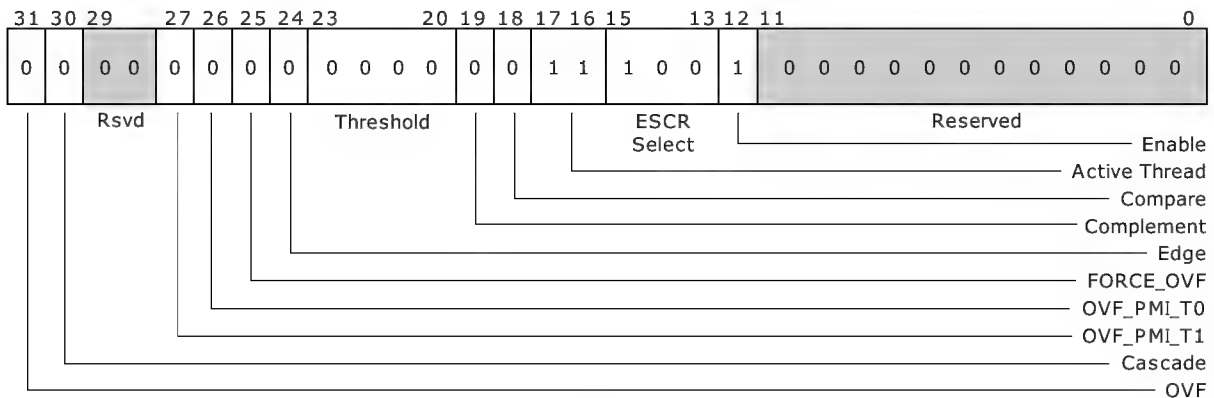
Event Mask field(ビット9～24)が設定されていると，イベント選択フィールドで選択されたイベントクラスから選択する．

Event Select field(ビット25～30)が設定されていると，カウントするイベントクラスを選択する．

ESCRの設定は，イベント選択フィールドでカウントするイベントのクラスを選び，そしてイベントマスクフィールドによって，そのイベントクラス中の一つ(あるいはいくつかの)イベントを選択する．

たとえば，リタイヤした分岐をカウントするとき，四つの異なるイベントを測定できる(分岐不成立予測成功/branch not taken predicted，分岐不成立予測失敗/branch not taken mispredicted，分岐成立予測成功/branch taken predicted，分岐成立予測失敗/branch taken mispredicted)．

ESCRはリセット時，0で初期化されている．ESCRのフラグやフィールドはwrmsr命令によりESCRに書き込むことに

〔図4〕Counter Configuration Control Register（CCCR）と記述例

Enable Flag, ビット 12
　設定によって，カウンタを有効にする．クリア時はカウンタは有効でない．リセット時はクリアされている
ESCR選択，ビット 13～15
　カウントするイベントを選択するために利用されるESCRを選択する
Active Thread, ビット 16～17
　00 —— どちらの論理プロセッサがアクティブでないときのみ計測する
　01 —— どちらかの論理プロセッサがアクティブのときのみ計測する
　10 —— 両方の論理プロセッサがアクティブのときのみ計測する
　11 —— 論理プロセッサがアクティブの時計測する
注：halt した論理プロセッサはアクティブではない
Compare flag, ビット 18
Complement flag, ビット 19
Threshold field, ビット 20～23
Edge flag, ビット 24
FORCE_OVR flag, ビット 25
　設定時，カウンタが増加するたびにカウンタオーバフローを強制する
OVF_PMI_T0 flag, ビット 26
　設定時，カウンタオーバフローが起きるたびにPMI（Performance Monitor Interrupt）を論理プロセッサ0へ送付する
OVF_PMI flag, ビット 27
　設定時，カウンタオーバフローがおきるたびにPMI（Performance Monitor Interrupt）を論理プロセッサ1へ送付する
Cascade flag, ビット 30
OVR flag, ビット 31

（a）CCCRのフォーマット：p.15-16

0x00039000 == ESCR select field == 4 == MSR_CRU_ESCR0
MSR_IQ_COUNTER0（== 0xC == 12）
MSR_IQ_CCCR0

（b）CCCRの記述例：p.15-57

〔図5〕性能カウンタ

よって設定できる．ESCRに書き込むだけではカウントを開始しない．単にカウントするイベントを選択するだけである．選択した性能カウンタ用のCCCRを設定する必要がある．CCCRはESCRを選択し，カウンタを起動する(**図3**，p.135)．

● **CCCR**

18個の性能カウンタは，それに対応する一つのCCCRをもつ．CCCRはイベントのフィルタリング，カウント，そして割り込みの生成などを制御する．下記はCCCR MSRのレイアウトを示す．CCCRもHyperThreadingのサポートにともなって，拡張されている．HyperThreadingをサポートしないPentium4プロセッサなどは，ビット27を利用できないので，注意が必要である．CCCRと記述例について，**図4**に示す．

● **性能カウンタ(Performance Counters)**

各性能カウンタは，**図5**のように40ビット長である．rdpmc命令により40ビットないしは下位32ビットを読むことができる．下位32ビット読み込みのほうが40ビット読み込みよりも速い．

rdpmc命令はどの特権モードでも利用できるが，CR4レジスタのPCE(Performance-monitoring Counter Enable)を0に設定することで，特権レベル0(カーネルモード)のみに制限することができる．

rdpmc命令はシリアライズされないので，カウンタを読むときに，前の命令が実行されるまで待つ必要はない．同様に，その後の命令もrdpmc命令の実行前に実行を開始するかもしれない．

rdmsrおよびwrmsr命令を利用しての性能カウンタの操作は，特権レベル0(カーネルモード)でのみ実行できる．カウンタを有効にする前にカウンタの値をセットする必要がある場合があるが，それはwrmsr命令を利用してカウンタに書き込むことによって行われる．オーバフローを設定する場合，2の補数の負の値を入力する．そしてカウンタは，設定した値から−1まで数え，そしてオーバフローする．Performance Counter，CCCR，ESCRの関連を**表2**に示す．

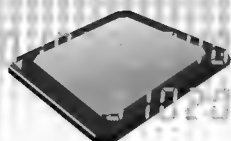
● **設定例**

イベントを計測する手順は下記のようになる〔読者の便宜のため参考文献1)の表15-4を**表2**に，表A-2を**表3**として掲載した〕．

1) 計測すべきイベントを選ぶ．ここでは例として，実行した命令数を計測するとする(instr_retired)
2) 参考文献1)の表A-1ないし表A-2(**表3**)よりinstr_retiredの項を見て，ESCRを選ぶ．ここでは，MSR_CRU_ESCR0を利用することにする．イベントによって選択できるESCRが異なるので注意する
3) 対応するCCCRとパフォーマンスカウンタを参考文献1)の表15-4(**表2**)から選ぶ．ここではMSR_IQ_CCCR0とMSR_IQ_COUNTER0を選ぶことにする(濃い網かけ部分)
4) 計測するイベントにESCRを設定する．イベントセレクトは2(instr_retired)，イベントマスクは0(Non-bogus，not tagged)とする(**表3**)
5) CCCRを設定する．ESCRの選択は4〔参考文献1)の表15-4のMSR_CRU_ESCR0の項目の数字〕とする
6) オプションとしてカスケード処理の設定，割り込み設定などをする
7) CCCRのEnableフラグを設定することによって計測を開始する

参考文献1)の表A-1ないし表A-2で，イベント名の列にはイ

〔表2〕**Performance Counter，CCCR，ESCRの関連**(出典：*Software Developer's Manual* table 15-4)

カウンタ			CCCR		ESCR		
名　称	No.	アドレス	名　称	アドレス	名　称	No.	アドレス
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0	7	3A0H
					MSR_FSB_ESCR0	6	3A2H
					MSR_MOB_ESCR0	2	3AAH
					MSR_PMH_ESCR0	4	3ACH
					MSR_BPU_ESCR0	0	3B2H
					MSR_IS_ESCR0	1	3B4H
					MSR_ITLB_ESCR0	3	3B6H
					MSR_IX_ESCR0	5	3C8H

〔表2〕**Performance Counter，CCCR，ESCR の関連**（出典：*Software Developer's Manual* table 15-4）（つづき）

カウンタ			CCCR		ESCR		
名 称	No.	アドレス	名 称	アドレス	名 称	No.	アドレス
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0	7	3A0H
					MSR_FSB_ESCR0	6	3A2H
					MSR_MOB_ESCR0	2	3AAH
					MSR_PMH_ESCR0	4	3ACH
					MSR_BPU_ESCR0	0	3B2H
					MSR_IS_ESCR0	1	3B4H
					MSR_ITLB_ESCR0	3	3B6H
					MSR_IX_ESCR0	5	3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1	7	3A1H
					MSR_FSB_ESCR1	6	3A3H
					MSR_MOB_ESCR1	2	3ABH
					MSR_PMH_ESCR1	4	3ADH
					MSR_BPU_ESCR1	0	3B3H
					MSR_IS_ESCR1	1	3B5H
					MSR_ITLB_ESCR1	3	3B7H
					MSR_IX_ESCR1	5	3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1	7	3A1H
					MSR_FSB_ESCR1	6	3A3H
					MSR_MOB_ESCR1	2	3ABH
					MSR_PMH_ESCR1	4	3ADH
					MSR_BPU_ESCR1	0	3B3H
					MSR_IS_ESCR1	1	3B5H
					MSR_ITLB_ESCR1	3	3B7H
					MSR_IX_ESCR1	5	3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0	0	3C0H
					MSR_TBPU_ESCR0	2	3C2H
					MSR_TC_ESCR0	1	3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0	0	3C0H
					MSR_TBPU_ESCR0	2	3C2H
					MSR_TC_ESCR0	1	3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1	0	3C1H
					MSR_TBPU_ESCR1	2	3C3H
					MSR_TC_ESCR1	1	3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1	0	3C1H
					MSR_TBPU_ESCR1	2	3C3H
					MSR_TC_ESCR1	1	3C5H
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0	1	3A4H
					MSR_FLAME_ESCR0	0	3A6H
					MSR_DAC_ESCR0	5	3A8H
					MSR_SAAT_ESCR0	2	3AEH
					MSR_U2L_ESCR0	3	3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0	1	3A4H
					MSR_FLAME_ESCR0	0	3A6H
					MSR_DAC_ESCR0	5	3A8H
					MSR_SAAT_ESCR0	2	3AEH
					MSR_U2L_ESCR0	3	3B0H
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1	1	3A5H
					MSR_FLAME_ESCR1	0	3A7H
					MSR_DAC_ESCR1	5	3A9H
					MSR_SAAT_ESCR1	2	3AFH
					MSR_U2L_ESCR1	3	3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1	1	3A5H
					MSR_FLAME_ESCR1	0	3A7H
					MSR_DAC_ESCR1	5	3A9H
					MSR_SAAT_ESCR1	2	3AFH
					MSR_U2L_ESCR1	3	3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0	4	3B8H
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH

〔表2〕**Performance Counter，CCCR，ESCRの関連**（出典：*Software Developer's Manual* table 15-4）（つづき）

カウンタ			CCCR		ESCR		
名　称	No.	アドレス	名　称	アドレス	名　称	No.	アドレス
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

〔表3〕**instr_retiredイベント**〔出典：参考文献1）の表A，一部抜粋〕

イベント名	イベントパラメータ	パラメータ値	定　義
instr_retired			1：This event counts instructions that are retired during a clock cycle. Mask bits specify bogus or non-bogus (and whether they are tagged via the front-end tagging mechanism). 2：The event count may vary depending on the microarchitecture state of the processor when the event is enabled. 3：The event may count more than once for some IA-32 instructions with complex uop flows and were interrupted before retirement.
	ESCR限定	MSR_CRU_ESCR0, MSR_CRU_ESCR1	
	ESCRごとのカウンタ番号	ESCR0：12，13，16 ESCR1：14，15，17	
	ESCRイベントセレクト	02H	ESCR[31：25]
	ESCRイベントマスク	Bit 0：NBOGUSNTAG 1：NBOGUSTAG 2：NBOGUSTAG 3：BOGUSTAG	ESCR[24：9]， Non-bogus instructions that are not tagged. Non-bogus instructions that are tagged. Bogus instructions that are not tagged. Bogus instructions that are tagged.
	CCCR選択	04H	CCCR[15：13]
	イベント固有の注意事項		1：The event count may vary depending on the microarchitectural states of the processor when the event detection is enabled. 2：The event may count more than once for some IA-32 instructions with complex uop flows and were interrupted before retirement.
	PEBSサポート	No	

ベントの名前〔たとえば，instr_retired(**表3**)など〕を示し，イベントパラメータの列には各種パラメータを示している．

▶ ESCR限定

イベントの測定に使用可能なESCRの一覧である．一般に，イベントをカウントするにはESCRが一つだけあればよい．

▶ ESCRごとのカウンタ番号

それぞれのESCRに対応付けられているパフォーマンスカウンタの一覧である．**表2**に，カウンタ番号と，カウンタとCCCRの名前が示してある．一般に，イベントをカウントするにはカウンタが一つだけであればよい．

▶ ESCRイベントセレクト

イベントを選択するためにESCRイベントセレクトフィールドに入れる値を示す．

▶ ESCRイベントマスク

カウント対象のサブイベントを選択するための値を設定する．パラメータ値の列には，0から始まるビット相対位置を入れる．ビット0は，ESCRのビット9に対応する．表記されていないビットはすべて0にセットしなければならない．

▶ CCCR選択

イベント定義をするために利用されるESCRを選択するためにカウンタと対応付けられているCCCRのESCR選択フィールドに入れる値を示す(この値はESCRのアドレスではなくて，**表2**のESCRのNo列の値である)．

▶イベント固有の注意事項

イベントについての補足事項を示す．

▶ PEBSサポート

イベントに対してPEBSをサポートするかどうかを示す(この情報は，表A-2に記載したAt-retirementイベントに対してのみ示される)．

▶タグ付けのために別のMSRが必要

イベントをカウントするのに，さらにMSRを必要とするかを示す(この情報は，表A-2に記載したAt-retirementイベントに対してのみ示される)．

▶性能カウンタの読み込み

rdpmc命令ないしrdmsr命令によって性能カウンタの値を読む．

▶イベントカウントの停止

性能カウンタを開始すると，無期限にカウントを続ける．カウンタがオーバフローすると，循環してカウントを続行する．カウンタが循環すると，OVFフラグがセットされ，カウンタがオーバフローしたことが示される．OVFフラグはスティッキーフラグであり，OVFビットが最後にクリアされてから1回以上カウンタがオーバフローしたことを示す．

カウンタを停止するには，そのカウンタのCCCRのEnableフラグをクリアする必要がある．

wrmsr命令を利用して，CCCRのEnableフラグをクリアする．

● **投機的実行とイベント**

Pentium4およびIntel Xeonプロセッサで利用されているIntel NetBurstマイクロアーキテクチャでは，多くの投機的実行を行っている．投機的実行とは，分岐命令のとき，分岐先を予測し，その予測にもとづいて命令をデコード・実行することである．分岐予測が失敗したとき，誤って実行した命令の結果はキャンセルされる．もし，性能カウンタがすべての実行された命令をすべてカウントするようにパフォーマンスカウンタが設定されている場合は，結果がコミットされた命令だけでなく，結果がキャンセルされた命令もカウントに含まれる．

こうした状況でのイベント計測の粒度を細密化するため，Pentium4/Intel Xeonプロセッサのパフォーマンスモニタリング機能は，イベントをタグ付けした後，コミットされた結果を表すこれらのタグ付けされたイベントだけをカウントする機能が提供されている．これを「At-retirementイベントカウント」と呼ぶ[注1]．

この機能により，次に示す二つのタイプのイベントを測定できる．

1) Non-retirementイベント．命令実行中に発生するイベント(たとえば，分岐リタイアメント，バストランザクション，キャッシュトランザクション)
2) At-retirementイベント．命令実行のリタイアメント時のイベント〔たとえば，tagging uOPs(micro operations)など〕．taggingによって実行パスのリタイヤの測定と，キャンセルされた実行パス(たとえば分岐予測に失敗したパス上の実行)の測定を区別できる

Pentium4およびIntel Xeonでは，次の三つの利用モデルがある．最初の二つはNon-retirementおよびAt-retirementイベントを測定できるが，三つ目の利用モデルでは，At-retirementイベントのサブセットしか測定できない．

1) イベントカウント

インターバルによってイベントを数える．

2) 精密でないイベントサンプリング(Non-precise event-based sampling)

カウンタがオーバフローしたときに割り込みがかかるように設定する．オーバフローを発生させるため，カウンタにあらかじめ値を設定しておく．カウンタがオーバフローすると，プロ

注1：Xeonは深いパイプラインをもつので，分岐先を予測して命令を投機的に実行するが，分岐予測が間違っていた場合は，分岐先での実行はキャンセルされる．taggingによってキャンセルされたイベント，キャンセルされなかったイベントを区別することができる．Non-retirementイベントでの測定では，コミットされなかった(キャンセルされた)命令によって発生したイベントも計測されてしまうが，At-retirementイベントによる計測だと，実際にリタイヤされた命令に関連するイベントのみを測定できる．Non-retirementイベントは，参考文献1)の表A-1を，At-retirementイベントは表A-2を参照してほしい．

〔図6〕DSセーブ領域

オフセット	Double Word Contents
0x00	BTS バッファベース
0x04	BTS インデックス：次のレコードへのポインタ
0x08	BTS 最大値
0x0C	BTS 割り込みしきい値(最大値以下)
0x10	PEBS バッファベース
0x14	PEBS インデックス
0x18	PEBS 最大値
0x1C	PEBS 割り込みしきい値(最大値以下)
0x20	PEBS カウンタリセット(下位32ビット)
0x24	PEBS カウンタリセット(上位8ビット)
0x30	予約 128バイト

〔出典：参考文献1)の図15-10〕

〔図7〕PEBSレコード

オフセット	Contents
0x00	Eflags
0x04	Linear IP
0x08	EAX
0x0C	EBX
0x10	ECX
0x14	EDX
0x18	ESI
0x1C	EDI
0x20	EBP
0x24	ESP

セッサはPMI(performance moniroring interrupt)を発生する．PMIに対する割り込み処理ルーチンは，RIP(return instruction pointer)を記録し，値を再設定し，カウンタを再実行する．たとえば，カウンタに－1000を設定しておけば，1000回ごとにオーバフロー割り込みが発生する．このRIPの分布を調べることによって，性能を分析することができる．

3) 精密なイベントサンプリング(Precise event-based sampling ――PEBS)

このタイプは，精密でないイベントサンプリングと似ているが，カウンタがオーバフローしたときのプロセッサの状態のレコードをメモリバッファに保存する部分が異なる．詳細は後述．

● **リタイヤメント時(At-Retirement)計測**

At-retirementカウントは，コミットされた命令に関するイベントだけあるいは投機的に実行し後に破棄された命令のイベントだけを計測できる．

IA-32 Intel Architecture Software Developer's Manual Volume 3 :System Programming Guideの表A-2からA-5にat-retirmentカウントを行うときのtaggingイベントが掲載されている．

用語の説明 Column

Bogus

分岐予測に失敗した実行パスにのっている命令なのでキャンセルされなければいけない命令ないしはuOP(マイクロ命令)．

Non-Bogus/Retire

コミットされた命令ないしはuOP．

したがって，命令はBogusないしはNon-Bogusになるが両方になるということはない．

たとえば，non-retirementイベントのカウントでL1キャッシュミスが多発していることを発見したとする．Bogus分岐(予測に失敗した分岐)上に多数のL1キャッシュミスがあるとすれば，分岐予測が失敗しにくいようにプログラムを書きかえることで，結果として，L1キャッシュミスを減らすことになる．

tagging

taggingは特定のイベントを検出したuOPをマーク付けする手段で，リタイアメント時にカウントすることができる．

reply

一般的な状況で性能を最大にするため，条件がすべて確実に満たされる前に，uOPを積極的にスケジュールして実行する．これらの条件がすべて満たされない場合には，uOPを再発行する必要がある．これをreplyという．replyはキャッシュミス，依存性違反および予測できないリソースの制約などが原因で生じる．

リタイヤメント時計測によって，コミットした命令に関わるイベントのみ，あるいは投機的に実行した後に取り消された命令に関わるイベントのみを計測できる．

● **At-retirementカウントの使用法**

Pentium4/Intel Xeonプロセッサは，指定されたイベントに遭遇したイベントおよびuOPをカウントできる．参考文献1)の表A-2に記載されているAt-retirementイベントの一部でタグ付けすることができる．タグ付け機能の一部でPEBSを利用できる．

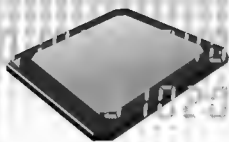
● **デバッグストア(DS ： Debug Store)**

デバッグストア(DS)は，Pentium4から導入された(**図6**)．この機能を利用すると，各種情報をカーネル空間のバッファに収集し，デバッグやチューニングに利用できる．これは，分岐レコードおよび精密なイベントベースサンプリング(PEBS ： Precise Event-based Sampling)の情報を収集するのに利用される．

IA32_MISC_ENABLE MSR(Model Specific Register)はパフォーマンスモニタリングとPrecise Event-Based Sampling (PEBS)の機能が有効であるかを示す．

● **PEBS(Precise Event Based Sampling)**

▶デバッグストア(DS)セーブ領域

PEBSレコード(**図7**)．性能カウンタがPEBS用に設定されると，カウンタがオーバフローするたびに，DSセーブ領域のPEBSバッファにPEBSレコードが格納される．このレコード

には，カウンタをオーバフローさせたイベントが発生した時点でのプロセッサの状態〔8個の汎用レジスタ(EAX，EBX，ECX，EDX，ESI，EDI，EBP，ESP)，EIP(論理アドレス)レジスタ，およびEFLAGSレジスタの値〕が格納される．そして，あらかじめ設定された値に自動的にリセットされ，イベントのカウントが再度開始する．

これはマイクロコードによって行われる(ハードウェアが自動的に実行する)ので，プロセッサの精密な状態の保存が可能となっている．DSセーブ領域に閾値を設定しておくと，それを越えるレコードが格納されたとき，PMI割り込みが発生するので，DSセーブ領域のデータをユーザー空間に退避するようなデバイスドライバを用意しておけば，DSセーブ領域を越えるレコードを取得できる．

● **イベントベースのサンプリングについて**

さて，上記の機能を利用するとイベントベースのサンプリングが可能になる．

性能カウンタ(PMC)は40ビットなので，2^{40}回イベントを計測するとオーバフローする．これを利用して，PMCにあらかじめ2の補数をセットしておけば，その回数ごとにイベントをサンプリングできるのである．たとえば，－100をPMCにセットすれば，100回目にオーバフローが発生するので，PEBSの機能(ハードウェアの機能)によって，イベントが発生したプログラムカウンタ(PC)，各種レジスタの値などが保存される．これをイベントベースサンプリングと呼ぶ．PEBSはAt-Retirementイベントのサブセットだけをサポートしている．

● **PEBSの設定**

PEBSの機能が利用できるかどうかは，CPUID命令で返されるDS機能フラグ(ビット21)で示される．このビットが存在する場合はPEBSの機能を利用できる．

PEBSにおいては，性能カウンタ(MSR_IQ_COUNTER4)しか利用できない．設定する方法は次のとおりである．

1) PEBS機能を設定する．DSセーブ領域のPEBSバッファベース/PEBSインデックス/PEBS最大値/PEBS割り込みしきい値/PEBSカウンタリセットの各フィールドに値を設定し，PEBSレコードバッファを設定する
2) PEBSをイネーブルにする．IA32_PEBS_ENABLE MSRのPEBSイネーブルフラグ(ビット24)をセットする
3) 参考文献1)の表A-2から表A-5に示されるように，PEBS用のESCR/MSR_IQ_COUNTER4/CCCRを設定する

おわりに

Pentium4/Intel Xeon以前のプロセッサはPEBSの機能がなかったため，PMCのオーバフローのたびに割り込みを発生させ，割り込みハンドラにより，各種レジスタ値を取得する必要があった．

しかしこの方法は前述のとおり，イベントサンプリングをする場合，ある回数ごとの例外処理ルーチンが起動され，実際のプログラムカウンタ(PC)や各種レジスタの情報を収集する時点では，例外処理ルーチンのレイテンシおよび上記のプロセッサの特性により，取得したPCの値は実際のイベントを発生させたPCよりかなり先を行っているという問題があった．

PEBSの機能によって，割り込みハンドラを起動するレイテンシやそのオーバヘッドが削減され，しかもイベント発生の正確な特定と，その時点での精密なコンテキストが入手できるようになった．

この機能を利用することによって，P6プロセッサなどでは不可能だった精密なサンプリングが可能になった．たとえば，L1キャッシュミスが多発している場所を精密に特定できるだけでなく，その時点での各レジスタの値も入手できるのでキャッシュミス時のプロセッサの状態を正確に理解することができる．

われわれはPEBSの機能にいちはやく注目し，それを利用しメモリプロファイリングツールを実装した．

次回は，われわれの実装とツールの利用方法について紹介する．

*　　　　　*

謝辞：メモリプロファイリングツールは平成14年度未踏ソフトウェア創造事業(プロジェクトマネージャー：喜連川優東京大学教授)「OLTP性能向上を目的としたメモリプロファイリングツール」として支援を受け，開発を行った．

参考文献

1) Intel, *The IA-32 Intel Architecture Software Developer's Manual Volume3: System Programming Guide*, Order Number 245472, 2002
2) Dean, J. et.al, " ProfileMe: Hardware Support for Instruction-Level Profiling on Out-Of-Order Processors ", *Proceedings of Micro-30*, December, 1997
3) Sprunt, B., " Pentium 4 Performance Monitoring Features ", *IEEE Micro*, July-August, 2002,
4) 吉岡弘隆，平成14年度未踏ソフトウェア創造事業，「OLTP性能向上を目的としたメモリプロファイリングツール成果報告書」
5) 吉岡弘隆，「Intel系(IA-32)プロセッサのパフォーマンスモニタリングファシリティを利用したメモリプロファイリングツール」，『第44回プログラミングシンポジウム』，箱根，2003年
6) 吉岡弘隆，「OLTP性能向上を目的としたメモリプロファイリングツール」，第14回『データ工学ワークショップ』，電子情報通信学会，2003年
7) メモリプロファイリングツール―― http://sourceforge.jp/projects/hardmeter/
8) よしおかひろたか，「未踏ソフトウェア奮闘記」，『日経ソフトウェア』，2003年7月号

よしおか・ひろたか　ミラクルリナックス(株)

音楽配信技術の最新動向

Ogg Vorbisのエンコードについて ——簡単なエンコーダの作成

第6回

岸　哲夫

今回も，前回に引き続いてOggVorbisのエンコーダの作成について説明します．前回はデコード処理をするプログラムである“vorbisfile”ライブラリを使いましたが，今回はそれに加えてエンコード処理を行う“vorbisenc”ライブラリを使います．

簡単なエンコーダの作成

エンコーダを作成するための開発環境にはLinuxとGNU Cを使用しますが，他の環境でもインクルードするヘッダを修正するだけで動作するはずです．

ただし，CPUによってはエンディアンの変更を行っている箇所を変更しなくてはなりません．そのため，インテルアーキテクチャのPC以外でプログラムする際は修正が必要です．

● ライブラリをインストールする

まず，必要なライブラリをインストールしてください．必要なものは以下のURLから入手できます．

```
http://www.vorbis.com/download.psp
```

ここから，libao，libogg，libvorbisの三つをダウンロードしてください．

RPMもありますが，tarballをコンパイルしたほうが自分が何をやっているかを意識できるのでよいと思います．

三つのライブラリは，以下のようにインストールします．

`tar zxvf hoge.tar.gz`で展開した後，できあがったディレクトリに移動します．そこで`./configure`を行ってください．その後，`make`，`make install`すれば完了です．

● プログラムを作成する

プログラムは，先に述べたとおりGNU Cで作成します．エンコーダプログラムは**リスト1**に示したとおりです．このプログラムは説明のために作成したものなので，詳細なチェックは行っていません．バグがあるかもしれませんが，個人の責任において使用してください．

コンパイルは，以下のように行います．

```
gcc -O3 -lvorbisfile -lvorbisenc enc_test.c
                                 -o enc_test
```

ここでリンクエラーが出た場合は，先に述べたライブラリのインストールがうまく行っていないと考えられるので，再度確かめてください．

使用法は，以下のとおりです．

```
enc_test < wav形式データ > OggVorbis形式データ
```

単純に標準入力をwavデータ，標準出力をOggVorbisデータに割り当てているだけです．

● 実際に起動してみる

筆者のコレクションから抜き出した音楽データをエンコードしてみます．

```
# ls -al *.wav
-rwxr--r--  1 root  root  47654818
                         5月 25 20:06 music.wav
#
# ./enc_test < music.wav > music.ogg
処理開始 2003/05/26 04:46:38
作成終了 2003/05/26 04:49:52
```

上記のように入力するとエンコードを開始します．Pentium4 2.66GHzの環境で3分半前後で終了します．元の曲は4分30秒でした．

```
# ls -al *.ogg
-rw-r--r--  1 root  root  7984159
                         5月 26 04:49 music.ogg
```

そして，このような大きさのoggファイルができあがりました．確認のためXMMSで聞いてみると，正常に再生されました．

非常に単純なプログラムですが，参考にしてください．このAPIはWindowsやMacOS 9，Mac OS X，BeOSそしてLinuxをインストールしたプレイステーション2にも対応しています．詳しくは，公式ドキュメントを参照してください．

ただし，このプログラムの処理は非常に遅いと思います．筆者が何らかの処理を間違っているのか，OggVorbisプロジェクトから配布されているものが最適化されていないのかはわかりません．

同じwavデータからmp3も作成してみました．比較のため同一の固定ビットレートで作成してみました．

```
# ls music.* -Al
-rwxr--r--  1 root  root  8647574
                         5月 26 04:57 music.mp3
```

〔リスト1〕enc_test.c

```
 1://
 2://
 3://このプログラムはwav形式のデータを
 4://                  OggVorbis形式に変換するものです
 5://
 6://  enc_test < music.wav > music.ogg  とコマンドを
 7://  入力してください
 8://
 9://  gcc -O3 -lvorbisfile -lvorbisenc enc_test.c -o enc_test
10:// でコンパイルできます
11://   なおこのプログラムは機能説明用です.
12://          使用する場合は個人の責任において使ってください.
13://           バグがあるかもしれません
14://
15:#include <stdio.h>
16:#include <stdlib.h>
17:#include <string.h>
18:#include <vorbis/vorbisenc.h>
19:#include <time.h>
20:#include <math.h>
21:
22:#define CON_READ    2048
23:#define BIT_RATE    256000
24:char readbuffer[CON_READ*4+44];
25:
26:int main(){
27:    int             eos =   0;
28:    int             i;
29:    int             result;
30:    vorbis_info      V_info;
31:    vorbis_comment   V_com;
32:    vorbis_dsp_state V_dsp;
33:    vorbis_block     V_blk;
34:    ogg_stream_state Ogg_Stream;
35:    ogg_page         Ogg_PG;
36:    ogg_packet       Ogg_Pac;
37:    time_t ct;
38:    struct tm *lst;
39:    char    tstr[1024];
40://処理開始
41:    fprintf(stderr,"処理開始 ");
42:    time( &ct );
43:    lst = localtime( &ct );
44:    strftime( tstr, 1024, "%Y/%m/%d %H:%M:%S¥n", lst );
45:    fprintf(stderr,tstr);
46:    vorbis_info_init(&V_info);
47:    if  (   vorbis_encode_setup_managed(&V_info,2,44100,BIT_RATE,BIT_RATE,-1)   )   exit(1);
48:    if  (   vorbis_encode_ctl(&V_info,OV_ECTL_RATEMANAGE_AVG,NULL)   )   exit(1);
49:    if  (   vorbis_encode_setup_init(&V_info)   )   exit(1);
50://コメントを付ける
51:    vorbis_comment_init(&V_com);
52:    vorbis_comment_add_tag(&V_com,"Title","kyokumei");
53:    vorbis_comment_add_tag(&V_com,"Artist","utau hito");
54:    vorbis_comment_add_tag(&V_com,"Album","hoge_hoge");
55:    vorbis_comment_add_tag(&V_com,"Year","2003");
56:    vorbis_comment_add_tag(&V_com,"Comment","test");
57://初期処理
58:    vorbis_analysis_init(&V_dsp,&V_info);
59:    vorbis_block_init(&V_dsp,&V_blk);
60:    ogg_stream_init(&Ogg_Stream,1);
61://ヘッダを出力する
62:    ogg_packet header;
63:    ogg_packet header_comm;
64:    ogg_packet header_code;
65:    vorbis_analysis_headerout(&V_dsp,&V_com,&header,&header_comm,&header_code);
66:    ogg_stream_packetin(&Ogg_Stream,&header);
67:    ogg_stream_packetin(&Ogg_Stream,&header_comm);
68:    ogg_stream_packetin(&Ogg_Stream,&header_code);
69:    ogg_stream_flush(&Ogg_Stream,&Ogg_PG);
70:    fwrite(Ogg_PG.header,1,Ogg_PG.header_len,stdout);
71:    fwrite(Ogg_PG.body,1,Ogg_PG.body_len,stdout);
72://エンコード開始
73:    while(!eos)
74:    {
75:        long i;
76:        long bytes=fread(readbuffer,1,CON_READ*4,stdin);
77:        float **buffer=vorbis_analysis_buffer(&V_dsp,CON_READ);
78:        //バイト並びの入れ替え
79:        for(i=0;i<bytes/4;i++)
80:        {
81:            buffer[0][i]=((readbuffer[i*4+1]<<8)|(0x00ff&(int)readbuffer[i*4]))/32768.0f;
82:            buffer[1][i]=((readbuffer[i*4+3]<<8)|(0x00ff&(int)readbuffer[i*4+2]))/32768.0f;
```

〔リスト1〕enc_test.c（つづき）

```
 83:         }
 84:         vorbis_analysis_wrote(&V_dsp,i);
 85:         while(vorbis_analysis_blockout(&V_dsp,&V_blk)==1)
 86:         {
 87:             vorbis_analysis(&V_blk,NULL);
 88:             vorbis_bitrate_addblock(&V_blk);
 89:             while(vorbis_bitrate_flushpacket(&V_dsp,&Ogg_Pac))
 90:             {
 91:                 ogg_stream_packetin(&Ogg_Stream,&Ogg_Pac);
 92:                 while(!eos)
 93:                 {
 94:                     result=ogg_stream_pageout(&Ogg_Stream,&Ogg_PG);
 95:                     if  (   result==0   )           break;
 96:                     fwrite(Ogg_PG.header,1,Ogg_PG.header_len,stdout);
 97:                     fwrite(Ogg_PG.body,1,Ogg_PG.body_len,stdout);
 98:                     if  (ogg_page_eos(&Ogg_PG))    eos =    1;
 99:                 }
100:             }
101:         }
102:     }
103://後処理
104:     ogg_stream_clear(&Ogg_Stream);
105:     vorbis_block_clear(&V_blk);
106:     vorbis_dsp_clear(&V_dsp);
107:     vorbis_comment_clear(&V_com);
108:     vorbis_info_clear(&V_info);
109:     fprintf(stderr,"作成終了 ");
110:     time( &ct );
111:     lst = localtime( &ct );
112:     strftime( tstr, 1024, "%Y/%m/%d %H:%M:%S¥n", lst );
113:     fprintf(stderr,tstr);
114:     return(0);
115:}
```

〔図1〕タグ情報

```
-rw-r--r--  1 root  root  7984159
                      5月 26 04:49 music.ogg
-rwxr--r--  1 root  root  47654818
                      5月 25 20:06 music.wav
```

ほんの少しですが，OggVorbisファイルのほうが小さくなっています．

プログラムの説明

それでは次に，今回作成したエンコーダプログラム`enc_test.c`のソースについて説明します．

● 15～20行目：インクルード文

18行目のヘッダファイルは，ライブラリをインストールしないとシステムに追加されないので注意してください．

● 22～23行目：定数の定義

● 24行目：読み込みのバッファの長さを設定

● 41～49行目：開始処理

パラメータの値を設定し，確保した構造体に設定しています．このプログラムでは，入力は2チャネル，サンプリングレートが44.1kHz（CD品質），ビットレートを256kbpsで設定しています．49行目の値を好みの値に変更してください．ただし，このプログラムでは入力したサンプリングレートを変更することはできません．

● 51～56行目：コメント処理

これは，IDタグに内容をセットする処理です（**図1**）．他の項目にもセットすることができます．日本語を使用する場合，使用する環境とCのコンパイル環境，およびCソースの漢字コードに注意してください．

● 58～60行目：初期化処理

他にも初期化すべき領域を初期化しています．

● 62～71行目：ヘッダ処理

OggVorbis形式のヘッダを出力しています．

● 73～102行目：エンコード処理

ここでは8236バイトを読み込み，バイト並びの変換を行い，エンコードを行っています．定数で設定した読み込み単位を変更すると効率が良くなるかもしれません．

● 104～114行目：後処理

領域の開放などを行っています．

以上の方法でエンコードを行っています．

きし・てつお

USB機器の相互接続性を保証する

USB Compliance Test の概要

林 徳義

USB2.0の発表に合わせて，新しいUSBのロゴが登場した．このロゴをつけてUSB機器を市販するには，相互接続性を保証するためにUSB Compliance Testをパスする必要がある．本稿ではUSB Compliance Testにおいてどのようなことをテストするのか，その概要を解説する． (編集部)

1. USB Certified Logoプログラムの衝撃

2000年11月15日，ComdexにおいてUSB-IF(USB Implementers forum Inc.)が『USB Certified Logoプログラム』を発表しました．この新しいブランド戦略は従来のロゴ〔**図1**(**a**)〕を廃止し，新しいロゴ2種類〔**図1**(**b**)〕を発表すること，そして新しいロゴを使用するにはCompliance Test(認証試験)に合格することを義務化しました．

このロゴプログラムは，USBのスピードを高速化させる新しい仕様，USB2.0と合うように発表されました．USB2.0は従来のUSB1.1の規格(ロースピードおよびフルスピード)に新たにハイスピードを追加した仕様です．ハイスピードは480Mbpsのデータレートを実現し，フルスピード(12Mbps)の40倍のスピードを実現します．フルスピードやロースピードの製品はCertifiedロゴで対応を表し，ハイスピード対応製品はHI-SPEEDロゴマーク付きで区別されます．

USB-IFがこのプログラムを開始するきっかけは，流通業者から強く要請されたのが原因といわれています．USB1.1時代，流通業者はUSB製品の相互接続性に起因する返品に悩まされていました．そのような背景からUSB2.0時代の幕開けにあたり，USB市場の安定と信頼を獲得するうえで，この新しいロゴプログラムが要請されたのです．

このプログラムは，ユーザーにとっては高品質の証として安心して製品を選択・利用でき，認証試験に従来から取り組んできた製造会社にとっては励みになるものだといえるでしょう．

〔図1〕USBのロゴ

(a) 廃止された古いロゴ

(b) 新しいロゴ(左がハイスピード対応，右がフル/ロースピード対応)

2. USBの基本仕様

● 接続形態

USBは，パーソナルコンピュータと周辺機器を接続する方法として1995年に登場しました．ホットプラグ機能を実現することによって，PC周辺デバイスが非常に使いやすくなりました．キーボードやマウスからプリンタ，スキャナ，PCカメラなどの広範囲なコンピュータ周辺機器をPCに簡単に接続できます．

典型的な接続形態を**図2**に示します．ホストはPCが1台のみで，USBケーブルで周辺機器と接続します．PCホストはルートハブといわれる内蔵ハブを含みます．

USBケーブルは最大5mまで拡張可能で，ハブは最大5段までカスケード接続できます．

PCホストからの信号をダウンストリーム，周辺機器からホストへの信号をアップストリームといいます．コネクタはダウンストリーム側をAコネクタ，アップストリーム側をBコネクタと呼びます．

● USBの信号線

USBは旧来のシリアルポートやパラレルポートに置き換わるようになりました．USBの利点の一つはPCと周辺機器の接続の柔軟性です．USBハブを経由すれば，ケーブル全長は最大30mまで拡張可能で，最大同時に127台まで接続できます．この柔軟性を実現したのは，電源ラインVbus，差動信号ライン

〔図2〕USBの接続形態

〔図3〕USBの信号線

〔図4〕ロースピードのダウンストリームパケットのようす

〔図5〕ロースピードのダウンストリームパケット解析結果

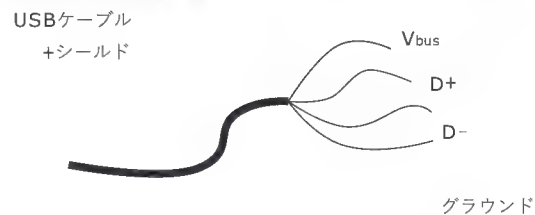
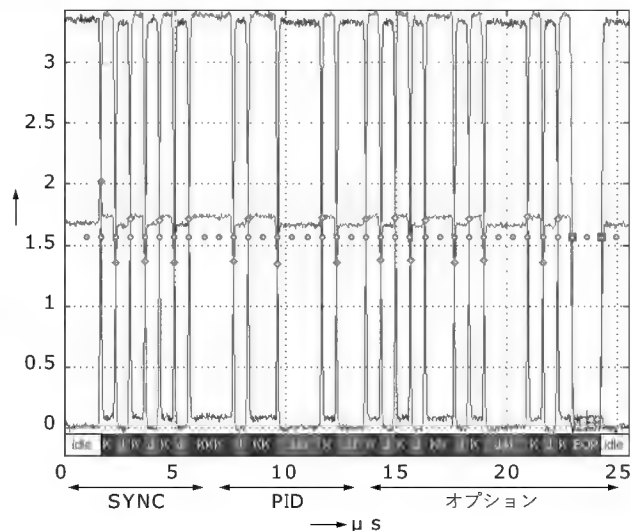
のD＋およびD－，およびグラウンドラインの4本のワイヤによるケーブルシステムです．ケーブルの中にはクロックラインはなく，SYNCフィールドをもったデータを通信させるEmbedded Clocking方式を取っています(**図3**)．

ハブの各ダウンストリームポートからは，最大で500mAの電流を供給可能です．これによりバスパワーの周辺機器はUSBケーブルからの電力に依存して駆動することが可能です．一方，伝統的なセルフパワーの周辺機器は独自にUSBとは別に電源ケーブルを接続することで動作できます．D＋，D－の差動伝送ラインはホスト，ハブ，周辺機器間の主要な情報キャリアの役目を果します．

● LSでのパケットのようす

このD＋，D－で流れる信号特性は フルスピード(以下FS)，ロースピード(以下LS)とハイスピード(以下HS)で大きく異なります．

図4に，LSでのダウンストリームパケットの通信のようすをオシロスコープで測定した画面を示します．測定はD＋，D－ともに独立のシングルエンドプローブで測定しています．D＋は緑色，D－が黄色です．信号振幅は約3.3Vであるのがわかります．

このパケットの中身を見ていきましょう．**図5**はオシロスコープで取り込んだデータをUSB-IFが開発したSignal Quality Test Toolのmatlab scriptで解析した結果です．通信パケット前後のIDLEステート，USBのステートのKステートおよびJステート，パケットの終了を示すEOPが読み取れます．IDLE状態はJステートで始まります．前のステートからの変化があるときはデータとして0，変化がない場合はデータ1が送られるNRZI codingでデータが転送されます．

パケットの最初の8ビットはKJKJKJKKで構成されており，SYNCビットになっています．続く8ビットは必ずPID(Packet Identifier)で構成されます．PIDにはSOF，IN，OUT，ACKなど10種類のパケットタイプが定義されています．**図5**のPIDは1001を示しているので，INパケットであることがわかります．このINパケットのようなTokenパケットの場合，PIDに続くエリアはアドレスおよびEndPoint，CRCが続きます．最後に差動信号の2ラインはともに0Vを示すSE0と呼ばれるステートを示し，パケットの終了を示すEOPで終わります．

● HSでのパケットのようす

図6はHSの連続したTestパケットを測定しているようすです．差動プローブを使って測定しています．信号振幅は約400mVです．

LSとの比較のために，HSのINパケットを同じくUSB-IFのmatlab scriptで解析したものを**図7**に示します．

明らかに大きく異なる点は，

1) SYNCビットが32ビット(最小12ビット)になっていること
2) IDLEステートはJ/Kステートでなく，SE0になっていること

3) EOPはbit stuffingなしのNRZ 0111111であること

これにより480Mbpsのデータ転送を可能にしています.

● USBのデータ転送種別

簡単に各データ転送スピードを比較します(**表1**). USB2.0は通信する対象により, 次の4種類のデータ転送をサポートしています.

▶ **Interrupt Transfer**

マウスやキーボードなどを対象としており, 外部イベントに対して早急な割り込み処理を必要とする周辺機器との通信に使用されます.

▶ **Bulk Transfer**

プリンタやスキャナのように, 大量なデータ転送が必要で, しかもデータの信頼性が必要な場合に使用されます.

▶ **Isochronous Transfer**

スピーカやビデオのように, 大量なデータ転送ではありながら, データの信頼性よりも時間どおりに通信されることが重要視される場合に使用されます.

▶ **Control Transfer**

初期に周辺機器を接続したときのenumerationやコマンド初期化時に使われます.

3. USB Compliance Testについて

● テストの場所

USBの適合試験はいくつかの方法で受けることが可能です.

1) USF-IFが主催するplugfestに参加する方法

アメリカで年4回, アジアで年1回開催されます. 今年は昨年に続きアジアでは台湾での開催が予定されていますが, 時期はまだ未定です.

2) 独立系テストラボにて受ける方法

日本ではXXCAL(http://www.xxcal.co.jp/)などで実施されています.

実際の認証の手続きについて説明していきましょう. 認証試験には大きく分けて三つのテストから構成されています.

1) デバイスフレームワークテスト

2) 電気特性テスト

3) 相互接続性テスト

● デバイスフレームワークテスト

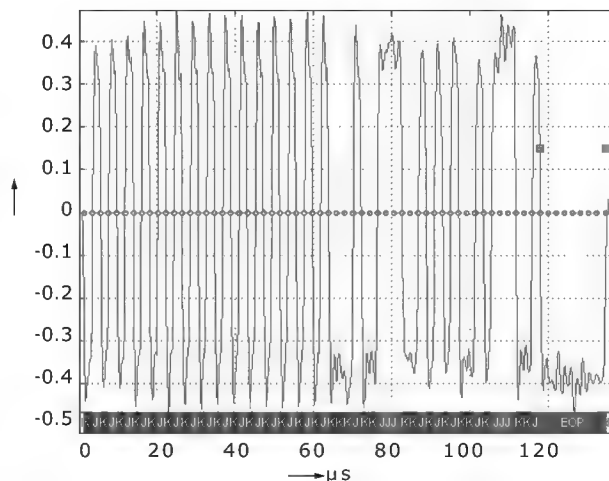
まず, 最初の難関がデバイスフレームワークテストです. デバイスフレームワークとはUSB2.0 Specification Chapter 9に記述されている仕様のテストであることからchapter 9テストとも呼ばれています.

USBのplugfestに参加した場合, check in前にテストされます. このデバイスフレームワークテストに落ちると, check inさえもできず, すごすごと帰るはめになるのでご注意ください.

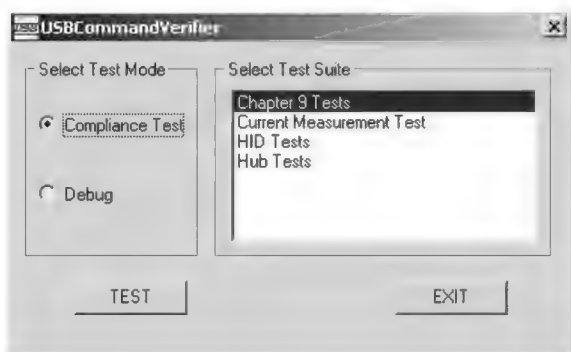
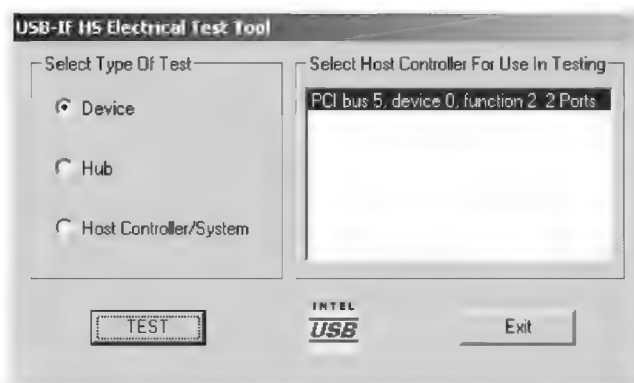
では, デバイスフレームワークとは何でしょうか? USB2.0 Specification Chapter 9を見ると, USBのデバイスは次の3層に定義されています.

- 最下層はバスインターフェースでパケットの送受信を行う
- 中間層はバスインターフェースとデバイスのいろいろな

〔**図6**〕**ハイスピードのダウンストリームパケットのようす**

〔**図7**〕**ハイスピードのダウンストリームパケット解析結果**

〔**表1**〕**スピード比較**

	ロースピード	フルスピード	ハイスピード
伝送レート	1.5Mbps	12Mbps	480Mbps
信号レベル	3.3V	3.3V	400mV
立ち上がり時間	75ns$<T_r<$300ns	4ns$<T_r<$20ns	$T_r>$500ps
計算上の信号成分の周波数帯域(BW=0.5/T_r)	6MHz	125MHz	1GHz

〔図8〕USB Command Verifier の画面

〔図9〕HS Electrical Test Tool の画面

endpoint 間のデータ経路を操作する
- 最上位の層はデバイスのファンクションを指す

デバイスフレームワークとは，この中の中間層の共通の属性と操作を定義したものです．デバイスフレームワークテストにはUSB CV（Command Verifier，**図8**）と呼ばれるプログラムが使われます（`http://www.usb.org/developers/tools` からダウンロード可能）．

USB CV はデバイスフレームワークテスト（chapter 9）だけでなく，Hub device class（chapter 11）および HID Specification についても行われます．

USB CV の注意点は，FS や LS デバイスをテストする場合でも HS のホストコントローラと HS ハブがないと動作できないことです．以前使われていた USB Check プログラムは，その機能性の制限から廃止され，より詳細なテストが実施されます．

また，正規にサポートしている OS は英語版の Windows 2000 および Windows XP です．ただ，Windows XP については日本語版でも動作している場合もあります．

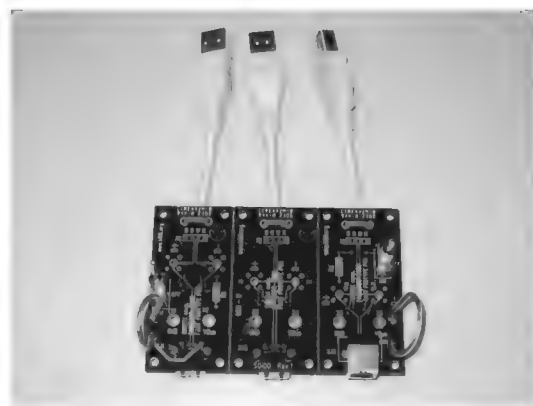
- 電気特性テスト

次に待ち構えている試験は，電気特性試験です．電気特性試験には多種の試験が待ち構えていますが，まず，FS/LS の試験から説明します．また，HS デバイスでも FS の試験は必ず通す必要があります．

FS/LS デバイス電気特性テストの内容としては，
- FS/LS Signal Quality Test
- In-rush Current Test
- Droop/Drop Test
- Backdrive voltage Test

から構成されます．

電気特性試験テストで使用するツールは，**図9**に示す HS Electrical Test Tool です．USB HS Electrical Test Tool（`http://www.usb.org/developers/tools` からダウンロード可能）．USB HS Electrical Test Tool の動作環境も USB CV と同じです．

また，これら FS/LS Test を実施するための治具を**写真1**に示します．この治具のことを Signal Quality Droop Drop ボード（略して SQiDD ボード）と呼んでいます．写真からわかるように三つの部分から構成されています．左端の部分は AA コネクタ（A コネクタのオスと A コネクタのメス）になっており，Signal Quality のプローブポイントと Inrush Current を行うためのスイッチおよび電流モニターループ，Droop/Drop 用のコネクタから形成されています．中間の部分も AA コネクタですが，Signal Quality のプローブポイントと Droop/Drop 用のコネクタだけです．右端は左端と同じですが，コネクタ形状が BB（B コネクタのオスと B コネクタのメス）になっています．

〔写真1〕FS/LS 用の治具 SQiDD ボード

▶ FS/LS Signal Quality Test

このテストはホスト，ハブ，デバイスすべてで実施しますが，ここではホストやハブではなく，デバイスの場合を例に説明します．

USB はその仕様で説明したように最大 5 段のハブを介し，しかもハブ間を最大 5m のケーブルで接続できることと規定されています．したがって，この最大構成で伝送パケットの信号品質が十分であることが必要です．従来からあるトラブルで，たとえば，3m の USB ケーブルで動作する機器が，5m のケーブルに変えたとたんに動作しなくなることがありました．これらの原因は信号品質に起因する問題で，相互接続性ではもっとも

〔写真2〕テストのようす

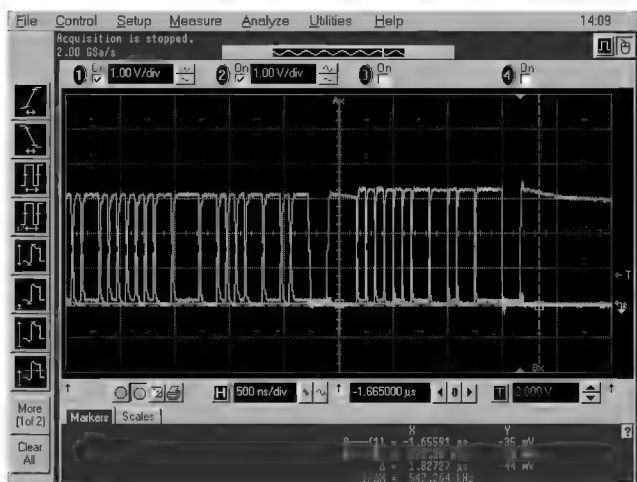
〔図11〕Signal Quality Testでのオシロスコープの表示

悩まされた例でした．

写真2にテストのようすを示します．写真の右側にハブが見えますが，テストでは五つのハブを接続してテストします．

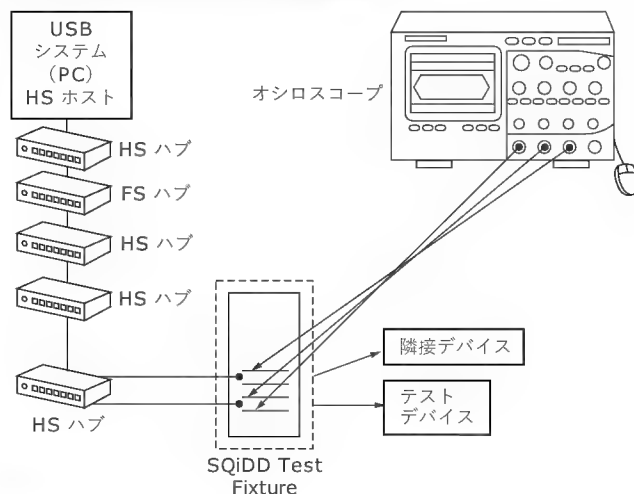
実際の接続図を**図10**に示します．隣接デバイスはテスト対象が出すパケットを，オシロスコープがトリガを取れるようにするためにテスト対象と同じスピードのデバイスを利用します．隣接デバイスがFSの場合にはD＋を，LSの場合にはD－をプロービングします．

PCホストでHS Electrical Test Toolを起動し，テスト対象のデバイスがenumerateされるのを確認してから，Loop device descriptorを選択して，PCホストとテスト対象との通信をループ状態に測定します．

オシロスコープには**図11**のような画面が表示されるので，デバイスのパケットがマーカ内にあることを確認します．

USB-IFのmatlab scriptを起動して解析すると**図12**のような結果がhtmlで表示されます．

よく聞かれる質問として，テスト結果項目にfailureと同時に

〔図10〕Signal Quality Testでの接続図

waiver grantedと表示されるということがあります．これは，規定値よりもテスト結果が若干悪い場合ですが，ただしそのときの認証試験としては合格扱いにするというメッセージになります．Waiverとは権利放棄を意味し，grantedとは認めることです．Waiverの値はその時々で変更されてきています．

FS/LS Signal Quality Test結果の内容を見ると，

- Signal eye
- EOP width
- Receivers: reliable operation on tier 6
- Measured signaling rate
- Crossover voltage range
- Consecutive jitter/Paired jitter

の項目からなっていることがわかります．matlab scriptはオシロスコープで取り込んだ差動データからクロスポイントを抽出して，まず信号のsignal rateの算出します．Signal rateから理想となる1クロックサイクルを算出し，各サイクルを重ね書きすることでアイパターンを作り，signal eyeのマスクパターンとの比較をします．また，同時に取り込んだクロスポイントの電圧値やジッタが総合評価されます．

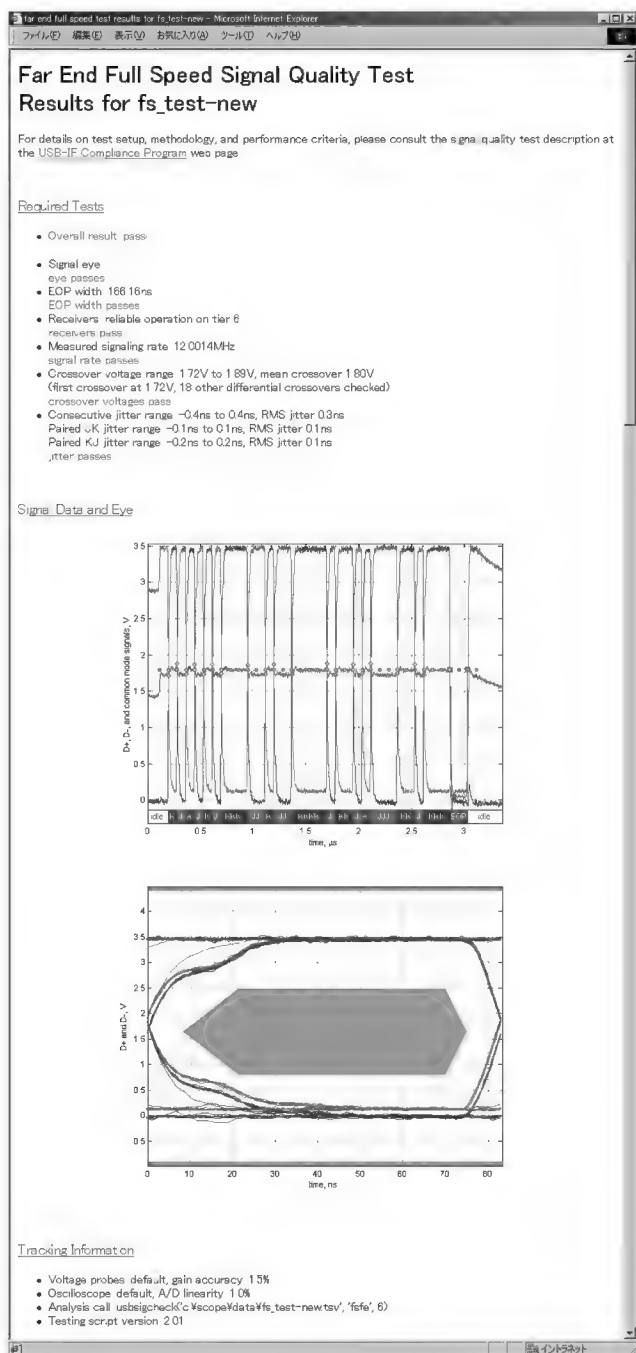
唯一，Receiversの項目だけは，試験が5段ハブを介してテストできたことを確認する項目で，測定波形から解析されるものではありません．matlab scriptは6以外の数字を入力すると動作しないようになっているので，かならず解析時には6を入れます．この数字が変わることで，アイマスクが変わることはありません．

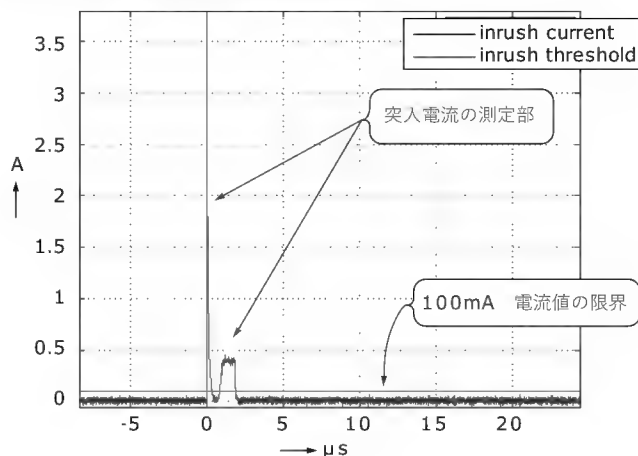
▶ Inrush current test

Inrush current testでは，USBコネクタが接続されたときの突入電流量を測定します．USBでは100mAを消費電流の限界として規定しているので，それを上回る量の電流を積算して，その合計が50μC以内であることを測定します．

このテストはバスパワーデバイスでfailすることが多く，セ

〔図 12〕Signal Quality Test での解析結果の html 表示

〔図 13〕Inrush current test の解析結果

ルフパワーデバイスではトリガさえかからないことも多くあります．測定には電流プローブで測定します．**図 13** にオシロスコープの測定値を USB-IF の matlab script で解析した結果を示します．

▶ Droop/Drop test

Droop/Drop test は，ホストおよびハブのダウンストリームポートの電圧降下について，その AC 特性および DC 特性について測定する試験です．ポートに 100mA および 500mA の負荷をかけ，他ポートの Vbus を測定します．

▶ Back Drive Voltage test

USB のアップストリームポートの Vbus は，いかなるときにも電流を供給していないこと，また，Vbus を供給する場合には D＋，D－の信号ラインのプルアップ抵抗に電力を供給していないことを確認する試験です．とくにセルフパワーデバイスで fail することが多いテストです．

● HS デバイスでのテスト

以上のテストに加えて，HS デバイスはさらに HS 電気特性試験が必要です．内容としては，

- High Speed Signal Quality Test
- Receiver Sensitivity and Squelch Test
- J/K Voltage Test
- Chirp Test
- Paket Parameter Test
- Suspend/Resume Test

から構成されます．とくに重要なのが HS Signal Quality Test と Receiver Sensitivity and Squelch test の二つです．

▶ HS Signal Quality Test

HS Signal Quality Test には**写真 3** の治具が使用されます．FS/LS の Signal Quality のテストが実際にホストとデバイス間の伝送パケットを解析するのに対して，HS Signal Quality Test では，テスト対象からあらかじめ定義された Test Packet を出力させて，治具の中にある 90 Ω終端抵抗に終端した波形をオシロスコープに取り込んで解析します．Test packet の出力制御は，FS/LS Test で使用した USB HS Electrical Test Tool で行います．ホストとデバイス間にはハブは必要ありません．

Signal Quality のテスト項目はほとんど同じです．matlab script からの結果例を**図 14** に示します．

▶ Receiver Sensitivity and Squelch Test

Receiver Sensitivity and Squelch Test は，テスト対象の受信感度の試験です．とくにハイスピード通信では 400mV とい

〔写真 3〕HS Signal Quality Test に使用する治具

うノイズに近い振幅信号を使用するので，その安定性や信頼性が重要です．

測定治具として，**写真 4** の治具が使用されます．HS Signal Quality Test 用の治具と外観上明らかに異なるのは，治具基板上に SMA コネクタが付いており，外部からの信号入力が可能になっている点です．

具体的なテストの接続図を**図 15** に示します．テスト手順としてはまず，ホストからテスト対象を SE0_NAK テストモードに制御します．この状態では，テストデバイスからは D＋，D－ともに 0V になるはずです．この状態で，外部からの信号源から USB の HS の IN パケットを入力します．

このテストでは信号源の振幅を 400mV 振幅から徐々に減らしていき，NAK パケットが返って来るところを観察します．外部信号源からのある振幅の IN パケットに対して，一部 NAK パケットを返さなくなるので，そのしきい値を記録します．これが 150mV 以下であることが必要です．さらに振幅を下げていくと，完全に NAK パケットを返さなくなります．そのしきい値を記録します．これが 100mV 以上である必要があります．ただし，現状の Test Proceddure には waiver として，それぞれ±50mV が認められています．

▶ J/K Voltage

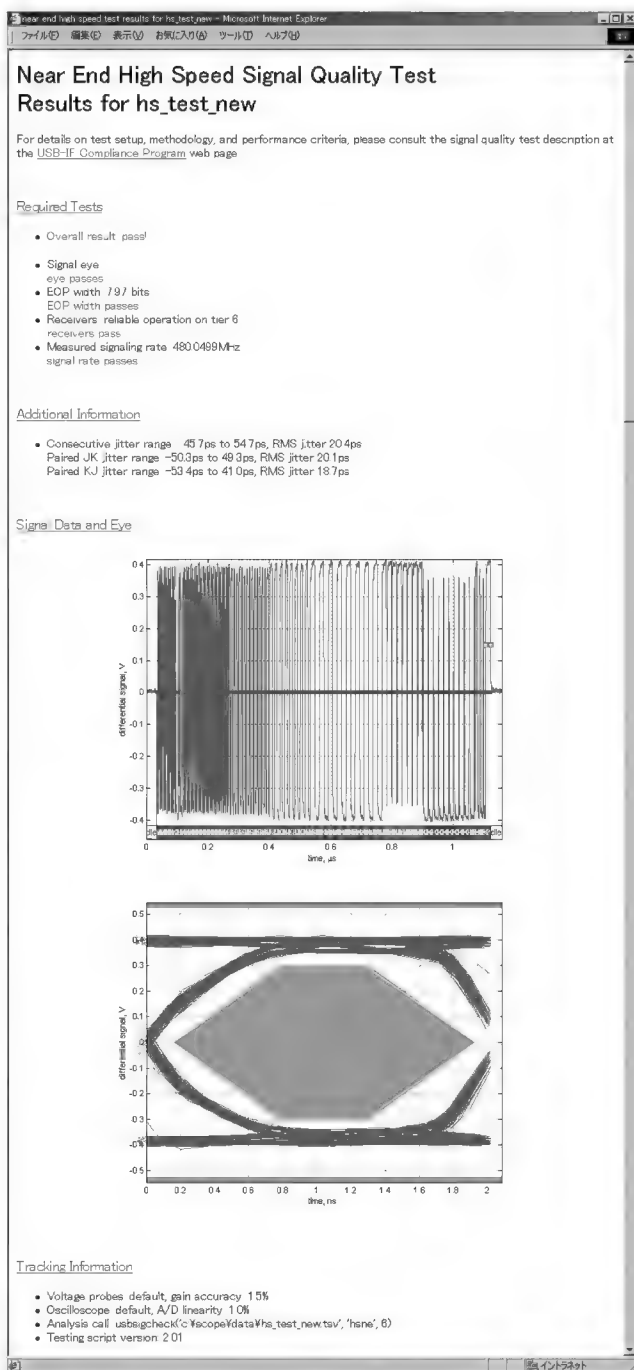
ハイスピードモードでの J/K ステートの静的な振幅を測定します．

▶ CHIRP Test

CHIRP テストは FS から HS にスピードを変える際のハンドシェイクを確認します(**図 16**)．

▶ Suspend/Resume

Suspend/Resume 時の振幅およびタイミング測定です．

〔図 14〕HS Signal Quality のテスト結果の html 表示

▶ Packet Parameter

Packet を構成する SYNC，EOP およびパケット間の間隔を測定します．

● 相互接続性テスト

最後は相互接続性のテストです．**図 17** のような Golden Tree と呼ばれる接続形態での enumeration の確認，ドライバのインストール，他のデバイスと PC の通信への干渉や PC との通信

〔写真 4〕Receiver Sensitivity and Squelch Test に使用する治具

中でのケーブル切断の影響などを調べます．とくにPCにインストールするドライバの性能が重要な要因になります．

まとめ

以上，USB Compliance Test について解説しました．これらのテストはすべてUSB-IFによりドキュメント化されており，容易に確認できます(http://www.usb.org/developers/docs からダウンロード可能)．

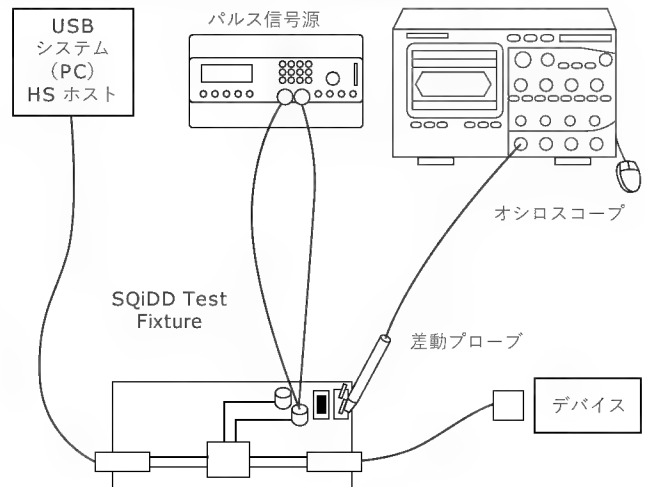
正規のTest Procedureには2種類あり，FS/LS用そしてHS用があります．ドキュメント自体はよくまとまっており，すべてのUSB開発者にとって，信頼性の高い製品開発を行ううえでの参考になるものです．

USB市場の安定と発展を切に期待します．

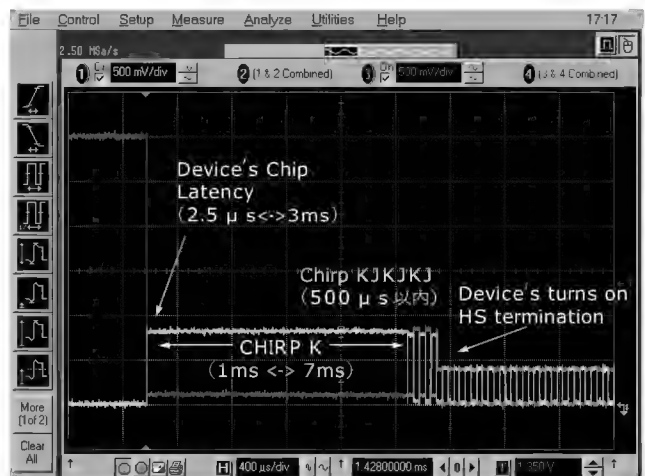
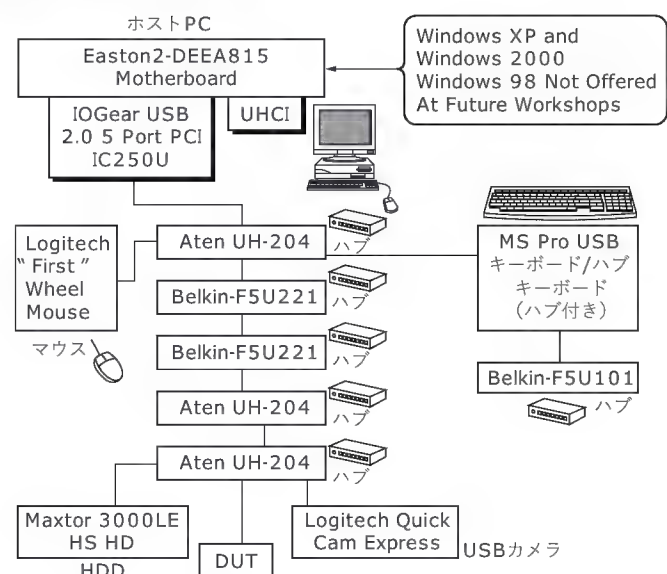
参考文献

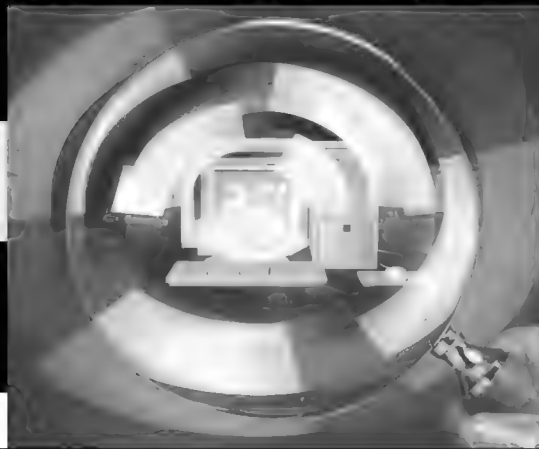
1) Universal Serial Bus Specification Revision 2.0(USB-IF)
2) Universal Serial Bus Implementers Forum Full and ロースピード Electrical and Interoperability Compliance Test Procedure Revision 1.2.1(USB-IF)
3) USB 2.0 High Speed Electrical Test Procedure version 1.0(USB-IF)
4) *USB Design by Example*, John Wiley & Sons INC
5) Universal Serial bus system architecture, MINDSHARE INC
6) " The eyes have it ", *EDN*. 6/13/2002

はやし・とくよし　アジレント・テクノロジー(株)　電子計測本部

〔図 15〕Receiver Sensitivity and Squelch Test の接続図

〔図 16〕CHIRP Test の結果

〔図 17〕Golden Tree の例

安価で高機能なUMLツール「Enterprise Architect」登場

日本語が使えるUMLツール最新比較

■ 酒井由夫

1 UMLとUMLツールの利用動向

オブジェクト指向設計において，対象となるシステムをモデリングするのに，UML（Unified Modeling Language）で描いた各種の図面は必要不可欠です．UMLについては，近年ITの世界を中心に普及が進んだせいか，関連書籍は都市部のちょっと大きな本屋に行けばいろいろな本が見られるようになりました．専門雑誌による紹介記事もたくさん見受けられます．しかし，

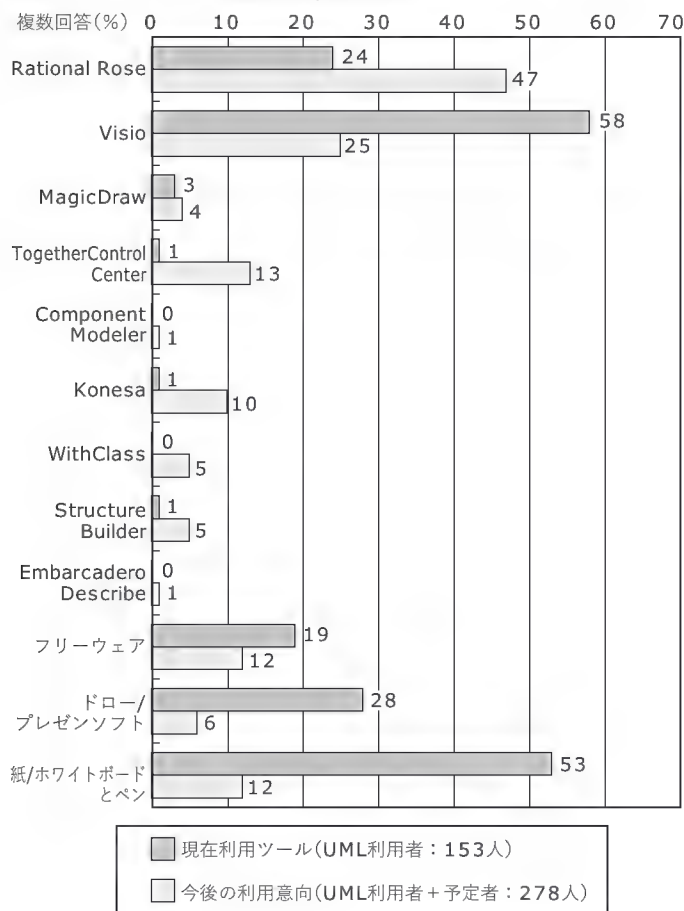
〔図1〕UML作成ツール利用状況/利用意向

実際にUMLで図を描くためのツールの普及率はどうでしょうか．**図1**に@IT（アットマーク・アイティ：ITエキスパートのための情報発信Webサイト，http://www.atmarkit.co.jp/）の第2回 読者調査結果（～UML利用の実態と課題とは～2002/10/5）で紹介された「UML作成ツール利用状況/利用意向」を示します．

図1を見ると，実際に使われているUMLツールのトップは「Microsoft Visio」で，以下「紙/ホワイトボードとペン」，「ドロー/プレゼンソフト」，「Rational Rose」，「フリーフェア」と続きます．これらのツールの全体における比率をみると「Microsoft Visio」と「紙/ホワイトボードとペン」の比率がそれぞれ50%を超え，とくに高いことがわかります．

なぜ「Microsoft Visio」のような汎用ドローイングツールや，「紙/ホワイトボードとペン」が，UML作成ツールとして多く使われているのでしょうか（紙/ホワイトボートとペンはツールとはいえないが……）？　**図1**の「今後の利用意向」もあわせて見ると，その理由を読み取ることができます．おそらくUMLの利用者は，Rational Roseのようなツールを使いたいけれど，Roseのような高価なUML専用のツールの購入を許可してもらえるほどまでには会社の上層部にUMLの存在が浸透していない，またはUMLツールを導入して業務効率が上がるという確信がUML利用者自身にない，UMLを使い始めたばかりで部下や同僚にその有用性を伝えるところまでマスタしていない，そのためUML専用ツールを買うところまで踏み切れないといった理由があると考えられます．

「紙/ホワイトボートとペン」は手軽に考えをまとめるには良いのですが，一度描いてしまった図を修正したり，ドキュメントとして残したり，再利用するといった面から見ると効率が良くありません．また，Visioはさまざまな図形をドローイングすることのできる優れたツールであり，Wordとの相性も良く，UMLの図をドキュメント化したり再利用することはできますが，いかんせんUML専用ツールではないため，かゆいところに手が届かないもどかしさを払拭できません．それにひきかえRational Roseは，UMLツールとして申し分ない機能を備えているのですが，価格が高いのが問題点であると筆者は考えます．

表1に，日本語が使えるUMLツールを書き出してみました．

〔表1〕日本語が使えるUMLツール

UMLツール名	製造元・販売会社	コメント
Rational Rose	IBM(Rational)	現時点でのUMLツールの標準．UMLモデリングのほか，構成管理，変更管理，開発プロセス，テストツールなどさまざまなRational製品との連携が可能
Rational Rose RealTime	IBM(Rational)	UMLモデリングに加え，RTOS対応のコードを自動生成できる(コード生成のパターンは限られるが，タスク分割をチューニングするスレッドマッピング機能を利用できる)
Visio2002	Microsoft	さまざまな図形を描画できるドローイングツール．電気系，機械系，ソフトウェア系など，さまざまなステンシルが用意されWordとの相性もよい．UML図面の描画はそのオールラウンドな図形描画機能をベースに一部拡張して実現している
Together	Borland	Borlandの設計分析ツール．GoFパターン，J2EEパターン，UIパターン，テストケースなど数多くのデザインパターンやテンプレートを搭載している
MagicDraw	(株)エッチ・アイ・シー	UMLドローイングツールとしての標準的機能は十分に満たしている．Javaベースなので OSを選ばない
Pattern Weaver	(株)テクノロジックアート	UMLドローイングツールとしての標準的機能は十分に満たしている．UMLのモデル部品やデザインパターンを登録して再利用できる．JavaベースなのでOSを選ばない
Konesa	(株)オージス総研	オージス総研は過去にRational Roseを販売していたが，日本における販売権が日本ラショナル(現IBM)に移ったことで現在はKonesaを取り扱っている．UMLドローイング機能はRational Roseと同等．サーバを立てることで複数ユーザーでの共同作業がスムーズに行える．Javaベース
Konesa-RealTime	(株)オージス総研 キャッツ(株)	UMLモデリングに加えて，ZIPC〔キャッツ(株)製〕の特徴である状態遷移図から状態遷移表への変換とRTOS対応のコードを自動生成することが可能．Javaベース
BridgePoint	(株)東陽テクニカ	モデルの作成からC/C++ソースコードの生成までを自動で行うことができる．高価だが，コード生成のルールをユーザーが独自に記述することが可能なので，組み込み機器への応用ができる(現実的にはBridgePoint専任の技術者が必要)
Enterprise Architect	スパークスシステムズ ジャパン	価格はRational Roseの10分の1以下ながら機能はRational Roseにひけを取らないといえる．2003年4月1日から日本語版がリリースされた
VEST-SAVER for Java	ヴェスト ソフトウェア(株)	PC上で動作する比較的安価な構造化分析手法のツール(VEST-SAVER)を作ったメーカーがリリースしたUMLドローイングツール．構造化分析モデルをオブジェクトモデルへ自動変換するナビゲーション機能をサポートしている
Rhapsody in C++	伊藤忠テクノサイエンス(株)	UMLドローイング機能はRational Roseと同等．各種周辺ツール(構成管理，テスト)との連携が特徴．いろいろなOSに対応したソースコードの生成が可能
WithClass	グレープシティ(株)	UMLドローイング機能はRational Roseと同等
IIOSS	IIOSS Project	オープンソースのUMLドローイングツール．ArgoUMLがベースになっている(無料)

※データは2003年5月現在のもの

日本語が使えるものだけでもこれだけあるので，英語でしか使えないものを含めると，かなりたくさんのUMLツールが世の中あるといえます．ただ，この中で圧倒的に知名度が高いのは，Visio2002とRational Roseです．

2 ソフトウェアエンジニアリングの表記法について

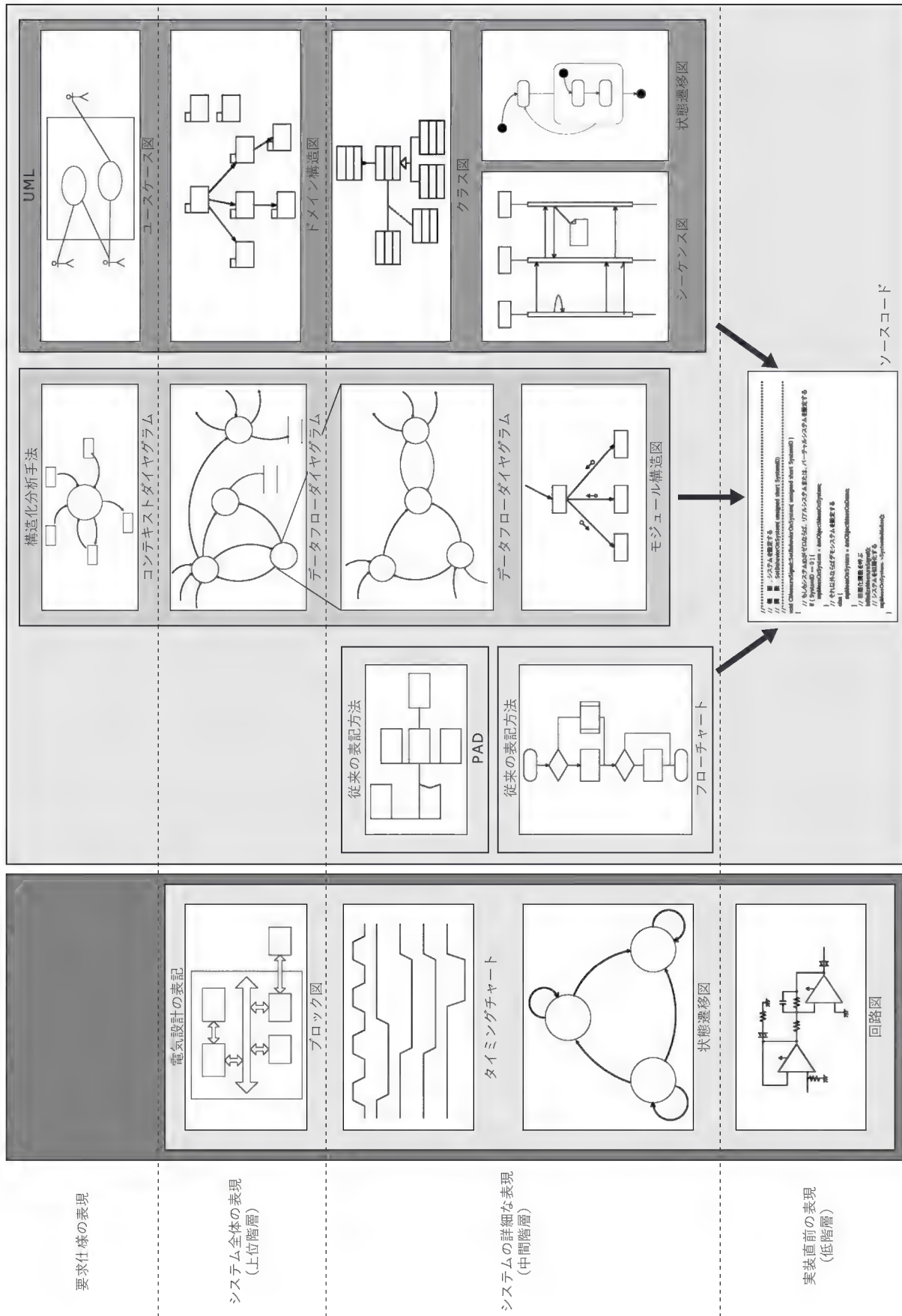
UML1.4は，広義にはあくまでも表記法であり，問題を解決するためのソリューションではありません[注1]．

UMLの図は，機械設計の世界でいえば機械図面，回路設計の世界でいえば回路図です．かつてソフトウェアエンジニアリングの世界では，フローチャートやPADなどが，ソフトウェアの中間的な階層を表現する表記法として使われていました．また，構造化分析手法のDFD(データフローダイヤグラム)も，ソフトウェアのシステムを階層的に表現する表記法です．フローチャートやPADはソースコードと同じく，ソフトウェアの表現の粒度が細かすぎるため，システム全体を見渡すのには向いていません．一方，DFDやUMLは，ソフトウェアシステムの全体から細部に至るまでのすべての工程を網羅しており，ソフトウェアシステムを上位層や中間層，下位層などをいろいろな角度からながめるのに便利です(**図2**)．

ただし，問題はこれらの表記法の普及率です．どんなに優れた表記法であっても，関係者の多くが表記法のルールについて理解していなければ，うまくコミュニケーションが取れません．また，一度描いたUMLの図をプロジェクトメンバやユーザーまたはクライアントと一緒にレビューし，よりよいモデルに洗練していくためには，表記法のルールだけでなく，ある程度自分自身でUMLを描いた経験がないと，実際には内容の濃い議

注1：UML 2.0では，UMLのモデルから直接ソースコードを生成する考え方Model Driven Architectureが取り入れられる．また，Rational Rose RealTime，Konesa-RealTime，BridgePoint，Rhapsody in C++など，RTOSに対応したソースコードをUMLモデルから生成するツールや，Xmodelinkなど UMLからSystemCを合成するツールもある．

〔図2〕表記方法の分析

エレクトロニクス
エンジニアリングでの表記法
ソフトウェア
エンジニアリングでの表記法
要求仕様の表現
システム全体の表現
（上位階層）
システムの詳細な表現
（中間階層）
実装直前の表現
（低階層）
電気設計の表記
ブロック図
タイミングチャート
状態遷移図
回路図
従来の表記方法
PAD
従来の表記方法
フローチャート
構造化分析手法
コンテキストダイヤグラム
データフローダイヤグラム
データフローダイヤグラム
モジュール構造図
UML
ユースケース図
ドメイン構造図
クラス図
シーケンス図
状態遷移図
ソースコード

論はできないでしょう．

ソフトウェアエンジニアリングの世界では，機械設計における機械図面や，回路設計における回路図に相当する汎用的で網羅性のある世界共通の表記法が，長い間存在しませんでした．そのため，エンジニア同士がソフトウェアの構造やふるまいについて議論するには，自分達だけのローカルなルールで作成したブロック図のようなものでソフトウェアシステムを表現するか，わかりにくいのを承知でソースコードを見ながら議論するしかありませんでした．ソースコードは，システム全体から見ると粒度が小さいうえ，プログラミング言語で書かれているため「読む」という行為なしでは内容を把握することができないという欠点があります．それに比べてエレクトロニクスエンジニアリングにおける回路図は実装直前の表現であるにもかかわらず，それぞれの部品の機能が直感的に理解しやすくできていて，回路図レベルで技術者同士が話をしてもとくに不自由はありません．

図2を見ればわかるように，UMLはソフトウェアシステムの要求仕様から，システム全体，システムの詳細な表現まで商品開発の全工程においてそのふるまいを表せます．さらに一部のUMLツールでは，実装直前の表現であるソースコードを自動生成することもできます[注2]．また，UMLは近いうちにISO（国際標準機構）に取り入れられるという動きもあるため，今後ソフトウェアの世界での標準的な表記法になる可能性があります．

3 UMLが普及する条件

表記法としてのUMLが日本で普及するためには，いくつかの条件が必要であると考えられます．

① 国際的な標準表記法であること
② 学習するための書籍やセミナーが豊富にあること
③ 安価で使いやすく，日本語での表記が可能なツールが存在すること

①は，Booch法を提唱したブーチ氏と，OMT法を提唱したランボー氏と，OOSE法を提唱したヤコブソン氏が協力して統合されたUMLができあがり，中立的な業界団体組織であるOMG（Object Management Group）に受け入れられた時点で，ある程度達成されたと考えられます．UMLがISOに組み入れられれば文句なく国際標準となるでしょう．

②については，「なぜこんなにUML関連の本が多いの？」と思うくらいの種類の書籍が本屋にあふれています．また，UMLに関するセミナーや，たとえばオージス総研が行っているUML認定試験制度や「オブジェクトの広場」などのWebフォーラムもあり，勉強しようと思えば教科書や情報は豊富です．

問題は③です．UMLは作成する図が多く，普遍的で抽象度の高いモデルも描くことができるため，一つ一つの部品の描き方や部品同士を結合するルールを理解するのが難しいという側面があります．そうなると，UMLを学習する以前にツールの使い方を覚えることが障壁となり，先に進むことが難しくなることもあるでしょう．したがって，UMLツールのヘルプはもちろんのこと，メニューやダイアログが日本語化されていないと，日本でUMLツールを広く普及させることが難しくなってしまいます．英語が必須条件になりつつある日本のビジネスシーンにおいても，母国語によるわかりやすい表現はすそ野を広くするためには必要です．また，ツール自体が高価だと，財布のひもを握っている人が先進的な考え方をもっていない場合，購入許可が下りないケースもあります．

安価で高機能なUMLツール Enterprise Architect

このような状況の中，2003年の4月1日にこの条件③を満たす安価で高機能なUMLツールの一つ，Enterprise Architectがスパークスシステムズ ジャパンより発売されました．このツールは2000年8月より，オーストラリアのスパークスシステムズ社より英語版が発売されていましたが，日本語版がリリースされ，日本のユーザーにとって一段と使いやすくなりました．

代表的なUMLツールである「Rational Rose」，「Visio2002」と「Enterprise Architect」の比較を**表2**に示します．それぞれのツールはさまざまなグレードをもっており，すべてのグレードをおしなべて比較することは難しいので，ノードロックスタンドアロン（パソコン1台につき1本のソフトウェアしか使えないようなしくみ）での使用を想定して，グレードを選定しています．より詳細な機能や仕様については，それぞれのツールベンダのWebページを見てください．まず注目すべきは価格で，Rational Rose Professional版は定価で¥252,000，Visio2002が¥69,800（オープンプライスのため実勢価格），Enterprise Architectが¥21,000です．

Enterprise Architect日本語版は，デスクトップ版/プロフェッショナル版/コーポレート版の3種類があり，デスクトップ版/プロフェッショナル版の1ライセンスの価格は，それぞれ¥13,000/¥21,000です．コーポレート版は2003年9月発売予定で，価格は未発表です．また，教育関係者用にアカデミック版価格が設定され，デスクトップ版/プロフェッショナル版の1ライセンスの価格は¥7,000/¥11,000です．エンジニアがポケットマネーで購入できる程度の価格であり，5人程度の小規模なプロジェクトであれば割引が適用されて総額¥95,000となり，社長決裁などなしに5本のソフトウェアを購入できます．また学生にとってもアルバイト代で購入できる範囲の価格だと思います．

Enterprise Architectを使うとき便利な機能を**表3**にまとめました（**図3**～**図5**も参照のこと）．

注2：現状ではソースコードの自動生成にはさまざまな制約条件があり，とくにリアルタイムシステムのような制約条件の厳しい組み込み機器にフィットした実行コードを出力させたい場合は，BridgePoint + DesignPointといったハイエンドなUMLツールを使わないと，完全に自動化することは困難である．

〔表2〕各種UMLツールの比較

比較項目	Rational Rose	Visio2002	Enterprise Architect
製造元または販売会社	IBM(Rational)	Microsoft	スパークスシステムズ ジャパン
グレード	Professional J Edition	Professional Edition	Professional
価格(ノードロック，スタンドアロン)	¥252,000	¥69,800(実勢価格)	¥21,000
アカデミック版	−−	¥30,000(実勢価格)	¥11,000
評価版の有無	×	○(¥1,050)	○(英語版)
サポート	○	△(回数限定)	△(Web&mail)
OS/動作環境	Windows/UNIX	Windows	Windows
UML各種図面描画機能	○	○	○
ユーザー別管理/共同作業	○	×	△※1
ソースとモデルの同期	×※2	×	○
構成管理	VisualSourceSafeとの連携可能※3	×	△※4
ドキュメント生成	html	html	html/RTF※5
リバースエンジニアリング	C++/Java※3	C++/VB/C#/VB.NET	C++/Java/VB/Delphi/C#/VB.NET
コード生成	C++/Java※3	×	C++/Java/VB/Delphi/C#/VB.NET

※1：プロジェクトファイルの複製と同期で対応．データベースによる管理はコーポレート版で可能
※2：Enterprise Editionで可能
※3：Windows NT版のみ
※4：XMLで出力し，構成管理ツールを使用することで差分管理できる
※5：Word互換のリッチテキストファイル

〔表3〕Enterprise Architectの特徴

●UML1.4に準拠しているすべてのUMLダイヤグラムを描ける(図3，図4，図5)
●図やUMLの要素をWindowsのエクスプローラ風のインターフェースで見ることができ，個々のユースケース，クラス，アクタなど，さまざまなUMLの要素や図を，格納されているパッケージ(Windowsのフォルダに相当)から別のパッケージへ簡単に移動できる→レビュー後に図を容易に移動，修正，追加できる
●パッケージの間をUMLの要素群が移動しても，UMLの要素間の依存，継承，集約など関連情報は失われず引き継がれる
●C++，Java，C#，VB，VB.NET，Delphiの生成と読み込み(リバースエンジニアリング)が可能→すでにあるプログラムソースを解析したり，作成したUMLのクラスからスケルトンのソースコードを生成できる
●XML(UML1.3 XMI1.1)形式でのUMLモデル入出力ができる→出力したXMLを構成管理することでコンパクトな差分管理が可能
●HTMLやWordと互換性のあるRTFファイル形式でのドキュメントを生成できるため，情報の共有や商品開発のドキュメントやクライアントへの提出資料としてそのまま使える
●テストダイアログを利用することで，作成したUMLの要素に対して単体，結合，システム，受け入れ，シナリオテストについて，「説明」，「入力」，「合格基準」，「実施状況」，「結果」などを記述でき，これらをシステム全体としてまとめたレポートをhtmlやRTFで出力できる→ソフトウェアライフサイクルプロセスの管理や，XP(eXtreme Programming)に利用可能
●プロフェッショナル版ではEnterprise Architectで作成したプロジェクトファイルをデザインマスタとし，マスタの複製を作成してメンバに配布し，追加・修正されたプロジェクトファイルの差分をデザインマスタに吸収する「複製の同期」機能を使うことができる→配布したプロジェクトファイルの変更をマスタと同期させることで分散開発をサポートする．コーポレート版ではSQLサーバなどとの連携が可能になる予定
●ユースケースの複雑度から，プロジェクトにかかる工数を見積もれる
●プロジェクトに関わるリソースの割り当てや顧客情報を入力できる→XP(eXtreme Programming)を運用する際に有効
●各種UML要素のプロパティダイアログやヘルプがきれいでわかりやすい

表3からはとても¥21,000のツールと思えないのですが，スパークスシステム ジャパンはそのWebページの内容から察するところ，個人や規模の小さい企業のユーザーにも利用可能なUMLのツールを提供し，UMLのすそ野を広げていきたいという考えがあるため，広告やカタログ，パンフレットの作成といった宣伝活動をいっさい行わず，サポートも電話やFAXでの質問は受け付けないことでこの価格を実現しているようです．ただ，電話やFAXでのサポートがないといっても，Web上でのQ＆Aや掲示板に多くの情報が掲載され，ツールに付属するサンプルプロジェクトやヘルプファイルも充実しているので，使用するにあたってとくに支障はありませんでした．

UMLとUMLツールの今後

UMLはISOに取り入れられる動きがあったり，かつXP(eXtreme Programming)を実践したいと思う人たちが増えたり，MDA(Model Driven Architecture)が話題になっていることから，ソフトウェアエンジニアリングの表記法としての中核になりつつあります．このような動きに加え，Enterprise Architectのような安価で高機能なUMLツールが出現してエンジニアや学生が個人的にUMLツールを購入できるようになったことで，その普及は加速されるのではないでしょうか．ソフトウェアの商品開発が日本国内だけでなく，グローバルな領域で行われるようになれば，ソフトウェアシステムの全体から詳細まで，エンジニア同士が国をまたがって議論する場面も増えてくるでしょう．そのようなとき，UMLのような国際標準の表記法があると，コミュニケーションを円滑に進めることができます．

UMLやUMLツールがソフトウェアエンジニアに浸透してくると，次はUMLでどのようなことができるのか，また，どんなメリットがあるのかという点に議論の焦点が移ってくると思

います．ただし，UMLが普及したからといって，すべてが事足りるというわけではありません．たとえば，UMLで描いたユーザー要求のユースケースについて実際にクライアント（ユーザー）を交えてレビューしてもらうためには，クライアントがUMLの表記法についての知識を持ち合わせている必要があります．それが可能かどうかはクライアントにまでUMLが普及するかどうかにかかっていますし，それが無理ならばユーザー要求を表現するためには，別の表現方法が必要になるでしょう．UMLのユースケース図と自然言語で書かれたシナリオだけがユーザー要求を表現する手段であるかどうかは疑問です．なぜなら，ユーザー要求の表現についてはMicrosoft PowerPointなどのOffice製品をはじめ，インターネットやイントラネットなどの情報基盤やWebページ，動画表現など，さまざまなITの発達によりグラフィカルで動きのある表現が可能になっているので，ソフトウェアシステムに対するユーザー要求は，もっと別の形で表現したほうがわかりやすい場合もあります．このようなユーザー要求の表現方法を共通化する必要性はとくになく，企画担当者やエンジニアがITを使ったわかりやすい表現手段をいかに使いこなせるかどうかが問題です．ユースケース図は設計工程の最上流に位置するものであるととらえたほうがよいでしょう．

ともあれ，UMLは今ブームなので流行に乗り遅れたくないと考えるソフトウェアエンジニアや，就職して即戦力になりたいと考える学生などにとって，Enterprise Architectのような安価で高機能なツールが発売されたことは朗報といえます．

また，比較的高価なUMLツールを扱ってきたベンダー各社は，Enterprise Architectのような安価なUMLツールが出現したことによって，今後オブジェクト指向設計の教育やソフトウェア開発者が直面している問題を実際に解決するためのコンサルテーション能力をアピールするようになるのではないでしょうか．また，UMLツールはおもにダイヤグラムを描くことを主目的とした安価なUMLツールと，UML2.0に準拠しモデルから直接実装可能なソースコードを生成するハイエンドUMLツールへの二極化が進むと予想されます．

参考文献・URL

1) アットマーク・アイティ（ITエキスパートのための情報発信Webサイト） http://www.atmarkit.co.jp/
2) スパークシステムズジャパン http://www.sparxsystems.jp/
3) （株）オージス総研 http://www.ogis-ri.co.jp/
4) 日本ラショナル（現IBM） http://www.rational.co.jp/
5) キャッツ（株） http://www.cats-hd.co.jp/
6) Microsoft Visio http://www.microsoft.com/japan/Office/visio/
7) 藤倉俊幸著，『リアルタイム/マルチタスクシステムの徹底研究』，CQ出版（株）
8) フレデリック・P・ブルックス,Jr. 著/滝沢 徹・牧野祐子・富澤 昇 訳，『人月の神話 新装版』，ピアソン・エデュケーション

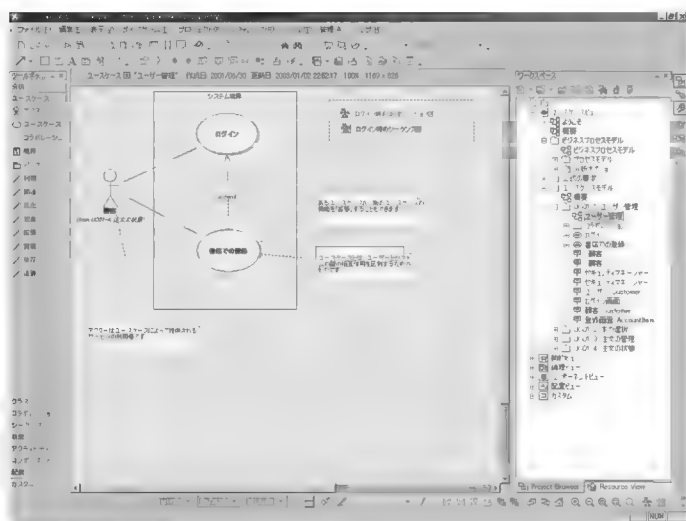
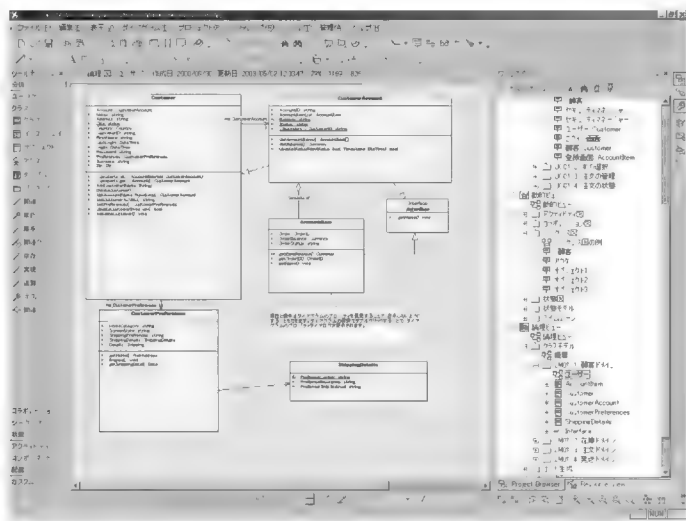
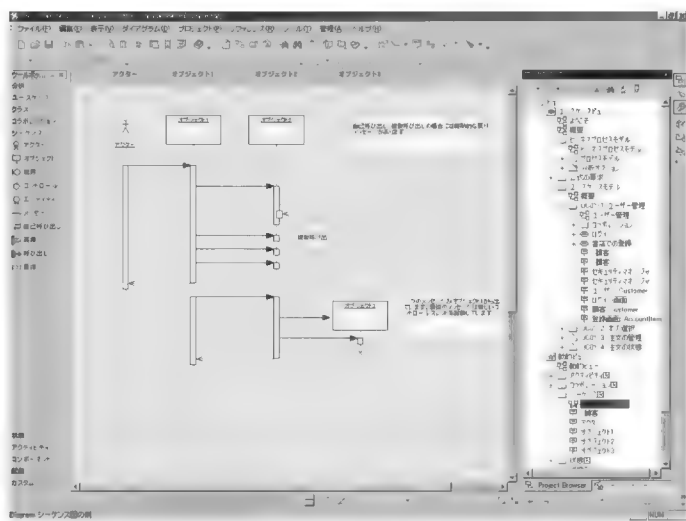
さかい・よしお SESSAME：組み込みソフトウェア管理者・技術者育成研究会 http://www.sessame.jp/

〔図3〕ユースケース図

〔図4〕クラス図

〔図5〕シーケンス図

CQ RISC評価キット

補足説明

SH-4PCI with Linux活用研究

SH-4 Linuxの割り込み処理とPCIの割り込み共有について

酒匂信尋

CQ RISC評価キット/SH-4PCI with Linux活用研究4(本誌2003年6月号)で，PCIボードをCN14に差し込むと割り込み処理ができなかったと説明した．今回は，オンボード上のLANコントローラと割り込みを共有する場合の処理について解説する．（編集部）

割り込みの処理方法としては，割り込みが発生したときに，割り込み禁止状態のまま割り込み処理をする方法と，割り込み要求フラグのクリアなどの必要最低限の処理を行ってから，別タスクに処理を引き継ぐ方法があります．

Linuxの場合も基本的には同じですが，ハードウェア構成を含めて，Linuxなりの作法があります．前回の活用研究(2003年6月号)で，割り込みを取得できなかったPCIスロット(CN14)を，割り込み処理の構造を調べながら動かしてみることにします．

1 Linuxの割り込み処理の構造

割り込み処理の流れの概要を**図1**に示します．

● レジスタ内容の保存

これは`linux/arch/sh/kernel/entry.S`に記述されている部分で，割り込みがかかる前に動いていたレジスタの内容を保存します．また，SHの場合，SR(ステータスレジスタ)のBLビットがセットされた状態での割り込み禁止になっているのですが，この状態での再度の例外の発生は許されないので，IMASKに0xfをセットしてBLビットをクリアにします．

● 当該割り込みのマスク

ここからは，`linux/arch/sh/kernel/irq.c`に記述されている部分で，どこからの割り込みであるのかを判別して，その割り込みをマスクします．

〔図1〕割り込み処理の流れ

Linuxの作法では，割り込み要因別にマスクできることが要求されます．SHの場合，内部のデバイスについては割り込みコントロールレジスタでマスクできますが，外部のデバイスでIRLに直結されている場合はマスクできないので，このしくみは実装できないと思われます．

しかし，SH-4用Linuxには解決策が用意されているので，次項で説明します．

● 割り込み禁止の解除

ここで，通常はデバイスドライバのハンドラを呼び出す前に，割り込み禁止を解除します．この状態でハンドラを呼び出すと，そのハンドラが動作中に，別のデバイスからの割り込みが発生すると，その割り込み処理が優先されることになります．

デバイスドライバのハンドラは，数百msの処理時間を有するハンドラも多々あります．その遅れは困るというような場合は，ここで割り込み禁止の解除をしないようにします．もっとも，ハンドラ内で数ms単位の時間を割り込み禁止のまま走るハンドラも多々あるので，本質的な問題解決にはならないかもしれません．実際，10ms周期で動くタイマ割り込みのハンドラは割り込み禁止状態のまま動作しますが，この周期で動ききれないことがあります．よく話題になるLinuxの時計の遅れは，このあたりの動きが原因ではないかと思われます．

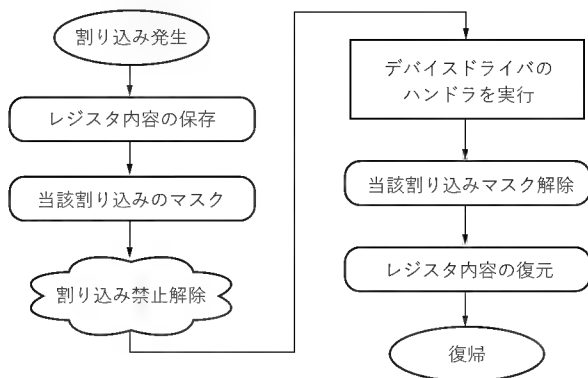
● デバイスドライバのハンドラを実行

ここで，当該割り込みに対応したハンドラを呼び出します．一つの割り込みに複数のハンドラが登録されていれば，それらのすべてを呼び出します．

● 当該割り込みマスクの解除

割り込み処理が終了したので，次の割り込み発生時に割り込みがかかるように，マスクを解除します．

● レジスタ内容の復元

割り込み発生時に動いていた状態に復帰するため，レジスタの内容を復元します．実際には，割り込みが発生する前の状態はカーネル内で動いていたのであれば，そのまま現状回復となりますが，アプリケーションが動いていた場合はスケジューラが呼び出されます．

2 割り込みのコントロール

● Lowレベルのハンドラ登録

割り込みのマスク/マスク解除といったLowレベルの処理は，linux/include/linux/irq.hにhw_interrupt_typeという構造体で定義されています．実際の処理は，SH用としてはlinux/arch/sh/kernel/irq_ipr.cに内部デバイス用，linux/arch/sh/kernel/irq_imask.cに外部デバイス用(IRL直結用)が用意されています．今回用いている評価キットの場合は，IRL0にAT互換機と同じ8259が付いている構成なので，linux/arch/sh/kernel/setuop_kzp01.cにkzp_irq_typeという名称で記述しています．

すでに説明したように，当該割り込みのマスクはdesc→handler→ack(irq)，マスクの解除はdesc→handler→end(irq)という記述でirq.cの中で呼び出されます．この登録は，内部デバイスの割り込みについてはirq_ipr.cで行われていますが，外部デバイスについては，setup_kzp01.cのinit_kzp_IRQ(void)で行っています．

IRLは個別の割り込みマスクができないのですが，irq_imask.cで記述されているIRL直結用は，SHの割り込みがレベル設定できることを利用して作成されています．割り込みのレベルが高いところでハンドラが動いた場合，それより低いデバイスの割り込みについては禁止状態と同じになってしまうので，使い方には注意は必要ですが，逆に使い方によっては処理を優先したい場合に効果的に使用することが可能になります．

余談ですが，Linuxも2.6からは優先度別のスケジューリングがサポートされるようですから，SHのもつ割り込みの優先度とスレッドの優先度管理をうまく使うことで，ライセンスが必要なリアルタイムUNIX系の商用OSと同等か，それ以上のリアルタイムOSとして改良することも，それほど難しくはないと容易に想像されます．

● デバイスドライバのハンドラ登録

ドライバのハンドラの登録には，request_irq()を使います．引き数はirq番号，ハンドラのアドレス，ふるまいを決めるフラグ，デバイス名称，デバイスIDとなります．

前回作成したドライバは，次のようになっています．

```
request_irq(irq_no, dio_drvr_interrupt,
                        SA_INTERRUPT, "dio", NULL);
```

フラグはSA_INTERRUPTとしています．これは割り込み禁止状態でデバイスドライバを呼び出す指定としています．IRQを共有する場合はSA_SHIRQとします．

3 割り込みの共有

前置きが長くなりましたが，いよいよ本題です．活用研究4で説明したドライバでは，PCIスロット3(CN14)にPCIボードを差し込むと割り込みを取得できませんでした．PCIスロット3はオンボードのEthernetのIRQと割り込みを共有しなければならないのですが，その点についての処理が抜けていたためです．

具体的には，PCIスロット3でDIOボードのPCIデバイスを検出した場合は，IRQを10ではなくオンボードEthernetと同じ7に設定します．

修正はsetup_kzp01.cで，差分を**リスト1**に示します．あとは，割り込み処理の説明からわかるように，ドライバのデバイスハンドラのフラグを割り込み禁止でなく，共有タイプに変更します．修正内容を**リスト2**に示します．

〔リスト1〕setup_kzp01.cの修正

```
--- setup_kzp01.c.old  Thu May  8 13:29:01 2003
+++ setup_kzp01.c Thu May  8 13:03:50 2003
@@ -190,6 +190,12 @@
          tmp = pci_cfgrd(bus_no,dev_no,0,PCI_VENDOR_ID);
          if( tmp == vendorid) {
           pci_cfgwr(bus_no,dev_no,0,PCI_COMMAND_MASTER,0x02000001);
+#if defined(CONFIG_PCNET32)
+          if(dev_no == 7) { /* CN14 */
+                pci_cfgwr(bus_no,dev_no,0,PCI_INTERRUPT_LINE, 0x07);
+                break;
+          }
+#endif
           pci_cfgwr(bus_no,dev_no,0,PCI_INTERRUPT_LINE, 0x0a);
           val = pci_cfgrd(bus_no,0x02,0,0x48);
           pci_cfgwr(bus_no,0x02,0,0x48, val | 0x00000003
                                              << 4*(dev_no - 5));
```

〔リスト2〕デバイスドライバの修正

```
--- dio_drvr.c.oldThu May  8 12:58:37 2003
+++ dio_drvr.c    Thu May  8 13:31:48 2003
@@ -111,7 +111,7 @@
    printk(KERN_DEBUG "dio_drvr :  io = 0x%04x,
                      irq_no = 0x%02x¥n",(int)io_start, irq_no);

-   if (request_irq(irq_no, dio_drvr_interrupt, SA_INTERRUPT,
+   if (request_irq(irq_no, dio_drvr_interrupt, SA_SHIRQ,
          "dio", NULL) != 0) {
     printk(KERN_DEBUG "dio_drvr : request_irq() error¥n");
     return (-ENODEV);
```

まとめ

従来の組み込み用のリアルタイムOSからLinuxへ移行する場合，割り込み応答時間の大きさやばらつきが，どうしても気になるところだと思います．ハードウェア設計時のちょっとした考慮とカーネルのちょっとした改良で，商用のUNIX系リアルタイムOSなみの割り込み応答性能を実現することもできるのですが，これらのノウハウについては，また機会があれば解説します．

さかわ・のぶひろ

UWB通信向けシミュレーションツール「UWB Entry Kit」の概要

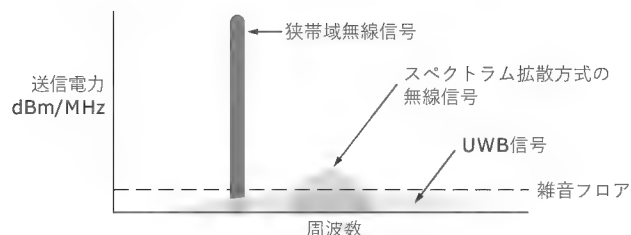
平良栄吉

2002年2月，米連邦通信委員会(FCC)が超広帯域(Ultra Wideband)無線システムの商用利用を許可したことから，にわかにUWB技術について関心が高まってきました．国内の動きはどうでしょうか．同じく2002年8月独立行政法人通信総合研究所(CRL：Communication Research Laboratory)で，UWB結集型特別グループが発足し，我が国における技術基準策定などが包括的に遂行されています．また，横浜リサーチパーク：YRPにUWB産学官コンソーシアムが結成され，UWB技術の商品化に関する産業界への貢献をおもな目的として，産学連携の共同研究が進められています(本誌2003年2月号特集記事参照)．

UWB技術の応用としてもっとも期待されているのは，無線通信システムでしょう．身のまわりのディジタル機器の高速な無線伝送(WPAN)への適用です．Intelは無線版USB2.0として開発を進めています．また家電メーカーでは，ビデオやオーディオといったストリームデータを配信するホームサーバへの適用を考えています．

筆者の会社〔(株)キューウエーブ〕は，UWB技術の通信システム適用への動きをとらえ，IEEE802.15で提案されているいくつかの技術，方式を短時間で理解，習得できる「UWB Entry Kit」を開発しました．これは，MATLAB/Simulink上で動作するシミュレーションキットです．

1 UWBとは何か

UWBは，Ultra Widebandの略で，文字どおり「超広帯域」を意味します．**図1**は無線通信信号の送信電力と周波数帯域の関係を示すイメージ図です．UWB信号はきわめて短時間のパルス(パルス幅が数十ps～数ns)で，放射電力スペクトル密度もきわめて低く(数pW～数十nW/MHz程度)さらに，比帯域幅が非常に大きいというのが特徴です．

- 比帯域幅＝帯域幅/中心周波数＞20％(－10dB帯域幅)
- 帯域幅：500MHz以上

が，米連邦通信委員会(FCC)のUWBの定義です(米DARPAの定義では比帯域幅＞25％)．

UWB信号は，次のような機能的特徴をもっています．

① 電力スペクトル密度がきわめて低い(雑音レベル以下)⇒既存の通信システムとの共存の可能性がある
② きわめて短い(ns単位)のパルスを利用⇒マルチパスに強い(高いパス分解能力)⇒高精度測距(数cm単位)が可能
③ キャリアなし，信号の放射時間がきわめて短い⇒小型・低消費電力のシステムを構築できる
④ 非常に広い帯域を占有(GHzオーダ)⇒超高速データ伝送(数百Mbps以上)⇒既存システムとの相互干渉は避けられない

米国では，UWBのこのような特徴を活かし，おもに軍事利用の分野で応用研究されてきました．しかし前述のように，2002年2月，FCCがUWB技術の商用利用を許可する規約を承認したのです．

〔図1〕無線通信信号の送信電力と周波数帯域の関係

出典：Xtreme Spectrum Inc.

2 UWB技術の応用例

UWBはその広帯域性のために，既存の無線システムとの干渉が懸念され，商用利用が認められていませんでしたが，米連邦通信委員会(FCC)が2002年2月14日に「超広帯域(UWB)」技術のマーケティングと運用に関する規約を承認しました．この規約は，引き起こす可能性のある干渉に基づいてUWB端末を3タイプに分け，それぞれ異なる技術規格と運用規定を設定しています．三つのタイプとは，①「画像システム」：地下の物体を探知するための地表貫通レーダ，壁中および壁の向こう側の物体を探知するレーダ，医療用の画像および監視機器，②「車両レーダーシステム」，③「通信システムおよび計測システム」です．

- 画像システム

画像システムとは，UWBを「レーダイメージング装置」に使

うというもの，といってもそのイメージの表示はテレビのようなものではなく，超音波システムのイメージ表示に近いものです．**図2**はTimeDomain社のRaderVisionです．この装置を使って壁の向こうの人質を探し出したり，人質を取った人物の居所を正確に割り出すことができます．ちなみに，TimeDomain社のRaderVisionは，15フィートと30フィートの二つのレンジを有しているようです．

● 車両レーダシステム

レーダ業界でのUWBの用途としては，前方の車や静止している物体への衝突を回避するのに役立つシステムがあります．また同じタイプのシステムが，衝突時のエアバッグの作動時間の改良に使われる可能性もあります．国内ですでに開発されている車載用ミリ波レーダは，「UWB」といえるほど比帯域幅は大きくないようです．

● 通信および計測システム

計測システムの例として，米Multispectral Solution Inc.(http://www.multispectral.com/)のGeolocation Systemを**図3**に示します．このシステムのおもな仕様は，最大出力4W，占有帯域幅400MHz，比帯域幅27％，計測距離：屋外見通し2km，屋内約100mとなっています．

通信システムはIEEE802.15.TG3aで，無線PAN(Wireless Personal Area Network)の高速通信方式として規格化されようとしています(本年度7月に規格が発表される予定)．提案されているおもな方式としては，次のようなものがあります．

① 直接拡散(DS-SS)方式：ソニー，米Xtreme Spectrum Inc.ほか

② 時間ホッピング(TH-PPM)方式：ST Microelectronics(スイス)，三菱電機ほか

③ マルチバンド方式：米Intel，米Time Domainほか

これらのうち，筆者の会社では，UWB技術の通信システム適用への動きをとらえ，これらの技術，方式を短時間で理解，習得できる「UWB Entry Kit」を製品化しました．以降でこれを紹介します．

〔図2〕レーダビジョン
(http://www.timedomain.com/radarvision/index.html)

3 UWB通信のシミュレーションを行う「UWB Entry Kit」

UWB Entry Kitは，研究者や技術者が広く利用している科学技術計算用ソフトMATLAB/Simulink上で動作するシミュレーションキットです．「UWB Blockset」，「シミュレーションモデルファイル」，およびこれらの解説書で構成されています．次のようなことが可能です．

▶ **UWB通信の送受信システム(シミュレーションモデル)を使用して，UWB通信の基本的なシミュレーション評価が行える**

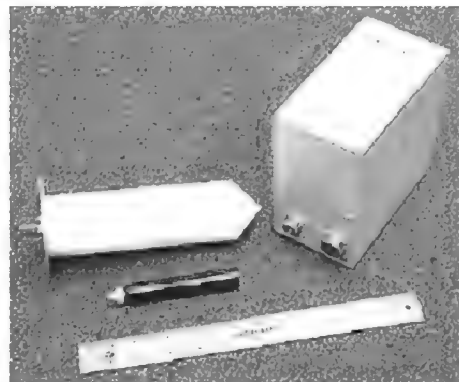
このシミュレーションモデルにおいて，送信，受信の過程をMATLAB/Simulinkの波形表示機能「Scope」を用いて視覚的に確認できます．これにより，スペクトル拡散通信，およびUWB通信の原理を容易に理解できます．

▶ **「UWB Blockset」には，UWB通信システムモデルを構築するのに便利な機能ブロックのほかにIEEE802.15提案のマルチパス伝送路モデルが含まれている**

IEEE802.15ではPER(Paket Error Rate)の求め方として，4タイプのマルチパス環境における100以上のマルチパス伝送路でシミュレーションした結果得られた上位90個のPERの平均値としています．このキットでは，IEEEが提案する4タイプのマルチパス環境下のマルチパス伝送路モデルを各100チャネル含むので，設計したシステムモデルのマルチパスシミュレーション評価ができます．

▶ **MATLAB/Simulinkの他のブロックと組み合わせて自由にシミュレーションモデルの拡張が可能**

「UWB Blockset」は，MATLAB/Simulinkの他のブロックとの接続を何ら制限していません．したがって，高度な知識をもち，自身で設計する回路を組み込みたい，またはシミュレーションモデルを拡張したいと思うユーザーは，自由にシミュレーションモデルの再構築が可能です．たとえば，バンドパス

〔図3〕UWB Precision Geolocation System Transceiver

〔表1〕UWB Blocksetの構成

カテゴリ	ブロック名	説　明
basic	UWB DownSampler	受信用ダウンサンプラ
	UWB Integrator	積分器
	UWB ModDelay	変調用パルス遅延器
	UWB PNGen	疑似乱数発生器
	UWB PulsGen	パルスジェネレータ
	UWB SeqDelay	タイムホッピング系列生成用パルス遅延器
	UWB Synthesizer	マルチバンド用周波数シンセサイザ
	UWB WaveFormer	マルチバンド用波形成形器
	UWB MonoCycle Pulse TemplateGen	テンプレート波形生成器
	UWB TimeHopping SequenceGen	タイムホッピング用シーケンス生成器
channel	UWB AWGN	AWGN発生器
	UWB MultiPath CM1～CM4	マルチパス伝送路モデル(CM1～CM4) IEEE 802.15提案モデル

〔表2〕DS-SS方式送受信モデルのおもな仕様

項　目	仕　様
変復調方式	BPSK同期検波方式
拡散符号長	31
誤り訂正	なし
チップレート	1GHz(1サンプル周期)
パルス幅	1ns(ただし，パラメータで変更可能)
パルス波形	3種類 ● モノサイクル波形 ● ガウシアンモノパルス ● 正弦波×ガウス窓

フィルタを追加してみたり，RAKE合成回路を組み込んでみるといったことが可能です．

本キットに含まれる機能ブロックの一覧を**表1**に示します．

次に，実際に「UWB Entry Kit」で何ができるのか，直接拡散方式(DS-SS)のシミュレーションモデルを例に，その中身を紹介します．

4 UWB Entry KitによるUWB通信システム(DS-SSモデル)

ここでは，DS-SS(Direct Sequence Spread Spectram)方式送受信モデルについて解説します．UWB Entry Kitでは，Simulinkの1サンプル時間(周期)を1ns(1GHz)として構成しています．DS-SS方式送受信モデルのおもな仕様を**表2**に示します．また，モデルのTopViewを**図4**に示します．

シミュレーションを実行することにより，「Scope」を使ってDS-SS送受信モデルの動作を確認できます．データの変調，拡散，逆拡散，データの復調を視覚的に確認できます．ユーザーは，確認したい部分の信号を自由に取り出し，Scopeに表示させることができます．

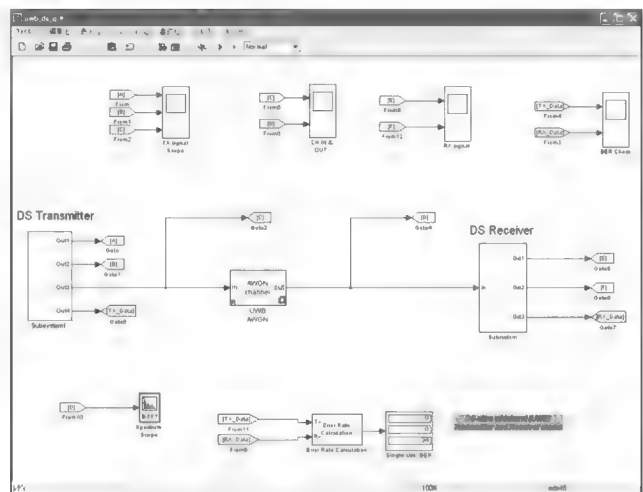
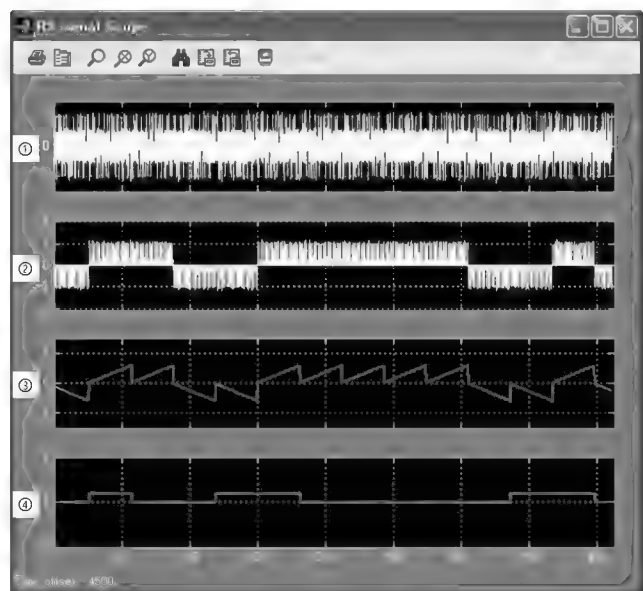
〔図4〕DS-SS送受信モデル

〔図5〕受信機内の出力波形

例として，受信機内の出力波形を**図5**に示します．DS-SSでは送信側で送信データに拡散符号系列を掛けてこれをパルスで出力しますが，受信側ではこの手順を逆にたどります．

図5はその過程を示したものです．①は受信信号，②は受信信号に拡散符号パルスを掛けた逆拡散波形，③はその積分出力，④は復調波形を示します．このように，動作過程を容易に確認できます．

ここではDS-SSモデルを示しましたが，本キットを使用すると，タイムホッピング方式の送受信モデルおよびマルチバンド方式の送受信モデルを容易に構築できます．

本キットに関する詳しい情報は，下記のURLで確認ください．

たいら・えいきち　(株)キューウエーブ

■ **UWB Entry Kitに関する問合せ先**：http://www.que-wave.com/

やり直しのための信号数学

第17回

DCTによる信号解析の基礎

三谷政昭

前回は，DFTからDCTを導き出すプロセスを紹介することで，DCTの基本的な考え方を理解してもらったつもりであるが，複雑な数式変形の醍醐味（?）を満喫された方もいれば，ウンザリされた方もおられるのではないだろうか（筆者自身も少なからず心配なのだが，ここが最初の踏ん張りどころ……かもしれない）．

今回は，DCTの物理的な意味付け，一般式を示すとともに，実務に直結させるための「DCTによる信号解析の基礎」を身につけてもらうことを主眼に，具体的数値例に基づき，わかりやすく解説する．（筆者）

DCTの一般式

いま，Nサンプルのディジタル信号$\{x_n\}_{n=0}^{n=N-1}$に対して，

$\{X_{\ell/2}{}^{(N)}\}_{\ell=0}^{\ell=N-1}$　：DFT（ディジタルフーリエ変換）

$\{C_\ell{}^{(N)}\}_{\ell=0}^{\ell=N-1}$　：DCT（ディジタルコサイン変換）

$\{S_\ell{}^{(N)}\}_{\ell=0}^{\ell=N-1}$　：DST（ディジタルサイン変換）

とするとき，三つの変換値の相互関係を**図1**に示す（詳細は，2003年5月号の第16回「DFTからDCTへの橋渡し」を参照）．

〔**図1**〕**DFT，DCT，DSTの相互関係**

	DFT ⁝ DCT	DST
直流($\ell=0$)	$X_0{}^{(N)}=C_0{}^{(N)}$	$S_0{}^{(N)}=0$
	DFT ⁝ DCT ⁝ DST	
交流($\ell\neq0$) ($\ell=1,2,\ldots,(N-1)$)	$X_{\ell/2}{}^{(N)}=C_\ell{}^{(N)}-jS_\ell{}^{(N)}$ $C_\ell{}^{(N)}=\frac{1}{N}\sum_{n=0}^{N-1}x_n\times\boxed{\sqrt{2}\cos\left\{\frac{(2n+1)\ell}{2N}\pi\right\}}$ $S_\ell{}^{(N)}=\frac{1}{N}\sum_{n=0}^{N-1}x_n\times\boxed{\sqrt{2}\sin\left\{\frac{(2n+1)\ell}{2N}\pi\right\}}$ 正規直交基底ベクトルの要素	

$$X_{\ell/2}{}^{(N)}=\frac{1}{\gamma_\ell}W_{2N}{}^{-\frac{\ell}{2}}\left[C_\ell{}^{(N)}-jS_\ell{}^{(N)}\right]$$

$$;\ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1)\quad\cdots\cdots(1)$$

ただし，
$$C_\ell{}^{(N)}=\frac{1}{N}\sum_{n=0}^{N-1}\gamma_\ell x_n\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\quad\cdots\cdots(2)$$

$$S_\ell{}^{(N)}=\frac{1}{N}\sum_{n=0}^{N-1}\gamma_\ell x_n\sin\left[\frac{(2n+1)\ell}{2N}\pi\right]\quad\cdots\cdots(3)$$

$$\gamma_\ell=\begin{cases}1 & ;\ell=0\\ \sqrt{2} & ;\ell\neq0\end{cases}\quad\cdots\cdots(4)$$

● 直流成分（$\ell=0$）の場合

まず，式(1)において$\ell=0$を代入すれば，

$$X_0{}^{(N)}=\frac{1}{\gamma_0}W_{2N}\left[C_0{}^{(N)}-jS_0{}^{(N)}\right]$$

となる．また，式(3)において$\ell=0$を代入すると，

$$S_0{}^{(N)}=0\quad\cdots\cdots(5)$$

であり，さらに式(4)より$\gamma_0=1$なので，

$$X_0{}^{(N)}=C_0{}^{(N)}\quad\cdots\cdots(6)$$

となる関係が得られる．つまり，$\ell=0$に相当する直流成分は，DFT値とDCT値とが同じ値をとるのである．

● 直流成分以外（$\ell\neq0$）の場合

式(1)の変形から始めよう．式(4)より，$\gamma_\ell=\sqrt{2}$なので，

$$X_{\ell/2}{}^{(N)} = \frac{1}{\sqrt{2}} W_{2N}{}^{-\frac{\ell}{2}} \left[C_\ell{}^{(N)} - jS_\ell{}^{(N)} \right]$$
$$= \frac{\sqrt{2}}{2} W_{2N}{}^{-\frac{\ell}{2}} \left[C_\ell{}^{(N)} - jS_\ell{}^{(N)} \right]$$
$$= \frac{1}{2} W_{2N}{}^{-\frac{\ell}{2}} \left[\sqrt{2} C_\ell{}^{(N)} - j\sqrt{2} S_\ell{}^{(N)} \right] \quad \cdots\cdots\cdots\cdots\cdots (7)$$

と表されることが容易に導き出される．

ところで，ある単一周波数の余弦波（あるいは正弦波）においては，最大値と実効値との間に，

（最大値）＝ $\sqrt{2}$ ×（実効値）$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$(8)

という $\sqrt{2}$ 倍の関係が成立することは知られている（**図 2**）．そこで式(8)の関係を考慮して，式(7)の物理的な解釈を試みよう．式(7)より，

$$X_{\ell/2}{}^{(N)} = \frac{1}{2} W_{2N}{}^{-\frac{\ell}{2}} \left[\overbrace{\sqrt{2}\ \underbrace{C_\ell{}^{(N)}}_{\text{実効値}}}^{\text{最大値}} - j\ \overbrace{\sqrt{2}\ \underbrace{S_\ell{}^{(N)}}_{\text{実効値}}}^{\text{最大値}} \right] \quad \cdots\cdots\cdots (9)$$

とみなせば，DCT 値 $C_\ell{}^{(N)}$（あるいは DST 値 $S_\ell{}^{(N)}$）は正弦波の実効値であり，その値を $\sqrt{2}$ 倍した値 $\sqrt{2}C_\ell{}^{(N)}$（あるいは $\sqrt{2}\,S_\ell{}^{(N)}$）は最大値に相当する量と考えられる．

たとえば，ディジタル信号の振幅成分のみを考えることにすると，式(7)の両辺の絶対値（振幅成分に相当）を採ればよいので，次のような関係が得られる（**図 3**）．

$$\left| X_{\ell/2}{}^{(N)} \right| = \frac{1}{2} \underbrace{\left| W_{2N}{}^{-\frac{\ell}{2}} \right|}_{1} \left| \sqrt{2} C_\ell{}^{(N)} - j\sqrt{2} S_\ell{}^{(N)} \right|$$
$$= \frac{1}{2} \times \sqrt{2} \times \sqrt{\left| C_\ell{}^{(N)} \right|^2 + \left| S_\ell{}^{(N)} \right|^2} \quad \cdots\cdots\cdots\cdots (10)$$

一例として，**図 4** のディジタル信号（山と谷を 1 個ずつ，1 周期分を含む）の DCT，DST の ℓ ＝ 2 に対する値を計算してみることにしよう．N ＝ 4 として，式(2)，式(3)より，

$$C_2{}^{(4)} = \frac{1}{4} \left[x_0 \left\{ \sqrt{2} \cos\left(\frac{2\pi}{8} \right) \right\} + x_1 \left\{ \sqrt{2} \cos\left(\frac{6\pi}{8} \right) \right\} + x_2 \left\{ \sqrt{2} \cos\left(\frac{10\pi}{8} \right) \right\} + x_3 \left\{ \sqrt{2} \cos\left(\frac{14\pi}{8} \right) \right\} \right]$$
$$= \frac{1}{4} \left[x_0 - x_1 - x_2 + x_3 \right] = \frac{1}{4} \left[2 - (-2) - (-2) + 2 \right] = 2 \quad \cdots\cdots\cdots (11)$$

$$S_2{}^{(4)} = \frac{1}{4} \left[x_0 \left\{ \sqrt{2} \sin\left(\frac{2\pi}{8} \right) \right\} + x_1 \left\{ \sqrt{2} \sin\left(\frac{6\pi}{8} \right) \right\} + x_2 \left\{ \sqrt{2} \sin\left(\frac{10\pi}{8} \right) \right\} + x_3 \left\{ \sqrt{2} \sin\left(\frac{14\pi}{8} \right) \right\} \right]$$
$$= \frac{1}{4} \left[x_0 + x_1 - x_2 - x_3 \right] = \frac{1}{4} \left[2 + (-2) - (-2) - 2 \right] = 0 \quad \cdots\cdots\cdots (12)$$

〔図 2〕最大値と実効値

〔図 3〕DFT，DCT，DST の振幅成分の相互関係

DFT　振幅成分（最大値）

$$\left| X_{\ell/2}{}^{(N)} \right| = \frac{1}{2} \times \sqrt{2} \times \sqrt{\left| C_\ell{}^{(N)} \right|^2 + \left| S_\ell{}^{(N)} \right|^2}$$

$\left(C_\ell{}^{(N)} - jS_\ell{}^{(N)} \right)$ の絶対値

||

振幅成分（最大値）の半分　振幅成分（実効値）

（負の周波数に対する振幅成分 $\left| X_{(-\ell/2)}{}^{(N)} \right|$ が残りの半分に相当）

〔図 4〕$C_2{}^{(4)}$ と $S_2{}^{(4)}$ の算出

〔図 5〕$X_{2/2}{}^{(4)}$（＝ $X_1{}^{(4)}$）の算出

〔図6〕 例題1

(a)

(b)

と算出される．よって，**図4**の信号の実効値は2であり，それを$\sqrt{2}$倍した値（＝2$\sqrt{2}$）は最大振幅となる．

一方，DFT値は，**図4**の1周期分の波形に対して計算するするわけで，$W_4=e^{-j\frac{2\pi}{4}}$とおけば，

$$X_{2/2}{}^{(4)}=\frac{1}{4}\left[x_0+x_1W_4{}^1+x_2W_4{}^2+x_3W_4{}^3\right] \quad \cdots\cdots(13)$$

が定義式である．**図4**の信号値を式(13)に代入するとDFT値は，

$$W_4=e^{-j\frac{2\pi}{4}}=e^{-j\frac{\pi}{2}}=-j$$

であることを考慮すれば，

$$X_{2/2}{}^{(4)}=\frac{1}{4}\left[2-j(-2)-(-2)+j2\right]=1+j \quad \cdots\cdots(14)$$

と計算される（**図5**）．他方，式(7)において，式(11)と式(12)の結果を利用すれば，$W_8=e^{-j\frac{2\pi}{8}}$なので，

$$\begin{aligned}&W_8{}^{-1}\left[\sqrt{2}C_2{}^{(4)}-j\sqrt{2}S_2{}^{(4)}\right]\\&=e^{j\frac{2\pi}{8}}\left[2\sqrt{2}-j0\right]\\&=\left\{\cos\left(\frac{2\pi}{8}\right)+j\sin\left(\frac{2\pi}{8}\right)\right\}\times 2\sqrt{2}\\&=\left\{\frac{1}{\sqrt{2}}+j\frac{1}{\sqrt{2}}\right\}\times 2\sqrt{2}=2+j2 \quad \cdots\cdots(15)\end{aligned}$$

となり，式(14)のDFT計算で算出した値(1+j)の2倍に一致することが確かめられる．つまり，$N=4$，$\ell=2$として，式(7)の関係，すなわち，

$$X_{2/2}{}^{(4)}=\frac{1}{2}W_8{}^{-1}\left[\sqrt{2}C_2{}^{(4)}-j\sqrt{2}S_2{}^{(4)}\right]$$

が成り立つことも検証できたことになるのである．

また，式(15)の複素数（直交座標）を極座標で表すと，

$$2+j2=2\sqrt{2}e^{j\frac{\pi}{4}}$$

となることから，最大振幅は2$\sqrt{2}$であることもわかる．

ところで，式(10)において√の値は，DCT値($C_2{}^{(4)}=2$)とDST値($S_2{}^{(4)}=0$)を代入して，

$$\sqrt{\left|C_2{}^{(4)}\right|^2+\left|S_2{}^{(4)}\right|^2}=2$$

と計算される．この値（＝2）は実効値なので，式(8)より実効値の$\sqrt{2}$倍した値（＝2$\sqrt{2}$）が最大振幅を示すことも理解される．

例題1

図6(a)，**(b)**に示すディジタル信号のDCT値$\left\{C_\ell{}^{(4)}\right\}_{\ell=0}^{\ell=3}$を求め，最大振幅値を推定せよ．

解答1

式(2)を適用して計算すればよい．いずれも最大振幅が2となるディジタル信号であることを，式(8)により各自で計算して確認してもらいたい（計算プロセスは省略）．

(1) $C_0{}^{(4)}=0$，$C_1{}^{(4)}=\sqrt{2}$，$C_2{}^{(4)}=0$，$C_3{}^{(4)}=0$ 〔**図6(a)**〕
(2) $C_0{}^{(4)}=0$，$C_1{}^{(4)}=0$，$C_2{}^{(4)}=\sqrt{2}$，$C_3{}^{(4)}=0$ 〔**図6(b)**〕

なお，参考のために$N=4$サンプルに対するDCTの数値計算式を以下に示しておく．

$$\begin{cases}C_0{}^{(4)}=\frac{1}{4}\left(x_0+x_1+x_2+x_3\right)\\C_1{}^{(4)}=\frac{1}{4}\left(1.3065x_0+0.5411x_1-0.5411x_2-1.3065x_3\right)\\C_2{}^{(4)}=\frac{1}{4}\left(x_0-x_1-x_2+x_3\right)\\C_3{}^{(4)}=\frac{1}{4}\left(0.5411x_0-1.3065x_1+1.3065x_2-0.5411x_3\right)\end{cases} \quad \cdots\cdots(16)$$

DCT値に基づくディジタル信号の再合成

いま，$N=4$サンプルのディジタル信号$\{x_n\}_{n=0}^{n=3}$について，そのDCT値を求めておく（**図7**）．式(2)あるいは近似計算の式(16)により，

$$C_0{}^{(4)}=1,\ C_1{}^{(4)}=2.0063,\ C_2{}^{(4)}=2,\ C_3{}^{(4)}=-2.9958 \quad \cdots\cdots(17)$$

となる．

ところで，4サンプルに対するDCTの正規直交基底ベクトル$\left\{\boldsymbol{\phi}^{(\ell)}\right\}_{\ell=0}^{\ell=3}$はそれぞれ，

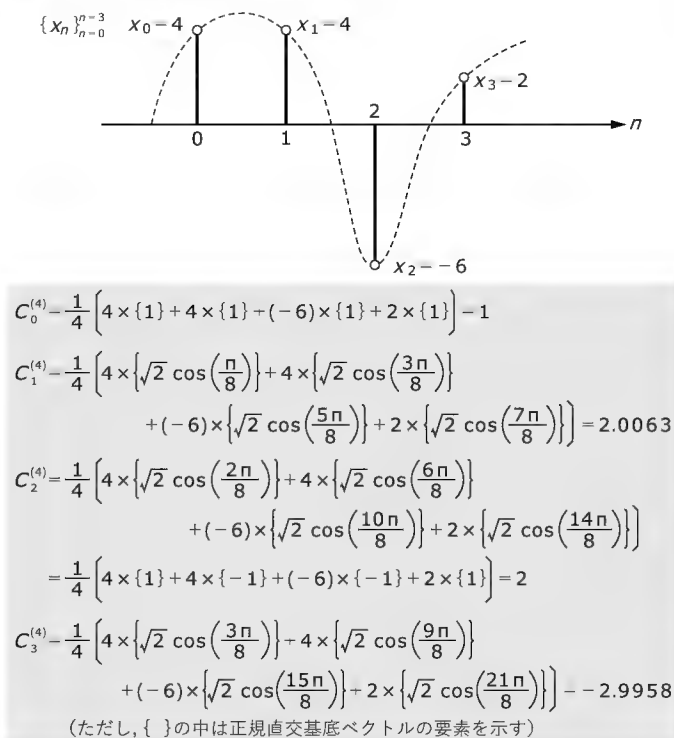
〔図7〕ディジタル信号とDCT値 $\left\{C_\ell^{(4)}\right\}_{\ell=0}^{\ell=3}$

$$C_0^{(4)}=\frac{1}{4}\Big[4\times\{1\}+4\times\{1\}+(-6)\times\{1\}+2\times\{1\}\Big]=1$$

$$C_1^{(4)}=\frac{1}{4}\Big[4\times\Big\{\sqrt{2}\cos\Big(\frac{\pi}{8}\Big)\Big\}+4\times\Big\{\sqrt{2}\cos\Big(\frac{3\pi}{8}\Big)\Big\}+(-6)\times\Big\{\sqrt{2}\cos\Big(\frac{5\pi}{8}\Big)\Big\}+2\times\Big\{\sqrt{2}\cos\Big(\frac{7\pi}{8}\Big)\Big\}\Big]=2.0063$$

$$C_2^{(4)}=\frac{1}{4}\Big[4\times\Big\{\sqrt{2}\cos\Big(\frac{2\pi}{8}\Big)\Big\}+4\times\Big\{\sqrt{2}\cos\Big(\frac{6\pi}{8}\Big)\Big\}+(-6)\times\Big\{\sqrt{2}\cos\Big(\frac{10\pi}{8}\Big)\Big\}+2\times\Big\{\sqrt{2}\cos\Big(\frac{14\pi}{8}\Big)\Big\}\Big]$$
$$=\frac{1}{4}\Big[4\times\{1\}+4\times\{-1\}+(-6)\times\{-1\}+2\times\{1\}\Big]=2$$

$$C_3^{(4)}=\frac{1}{4}\Big[4\times\Big\{\sqrt{2}\cos\Big(\frac{3\pi}{8}\Big)\Big\}+4\times\Big\{\sqrt{2}\cos\Big(\frac{9\pi}{8}\Big)\Big\}+(-6)\times\Big\{\sqrt{2}\cos\Big(\frac{15\pi}{8}\Big)\Big\}+2\times\Big\{\sqrt{2}\cos\Big(\frac{21\pi}{8}\Big)\Big\}\Big]=-2.9958$$

(ただし,{ }の中は正規直交基底ベクトルの要素を示す)

$$\boldsymbol{\phi}^{(0)}=\{1,1,1,1\} \quad \cdots\cdots(18)$$

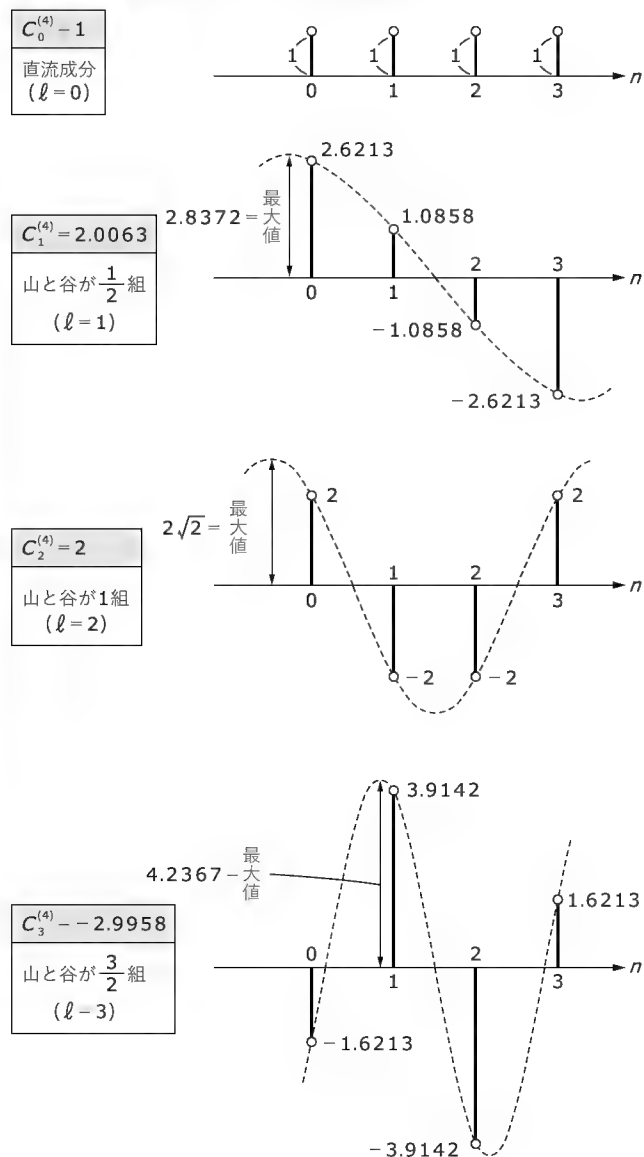
$$\boldsymbol{\phi}^{(1)}=\Big\{\sqrt{2}\cos\Big(\frac{\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{3\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{5\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{7\pi}{8}\Big)\Big\}$$
$$=\{1.3065,\ 0.5411,\ -0.5411,\ -1.3065\} \quad \cdots\cdots(19)$$

$$\boldsymbol{\phi}^{(2)}=\Big\{\sqrt{2}\cos\Big(\frac{2\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{6\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{10\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{14\pi}{8}\Big)\Big\}$$
$$=\{1,\ -1,\ -1,\ 1\} \quad \cdots\cdots(20)$$

$$\boldsymbol{\phi}^{(3)}=\Big\{\sqrt{2}\cos\Big(\frac{3\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{9\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{15\pi}{8}\Big),\ \sqrt{2}\cos\Big(\frac{21\pi}{8}\Big)\Big\}$$
$$=\{0.5411,\ -1.3065,\ 1.3065,\ -0.5411\} \quad \cdots\cdots(21)$$

であり,これらの正規直交基底ベクトルのDCT値による線形結合としてディジタル信号 x は表される.すなわち,

$$\boldsymbol{x}=C_0^{(4)}\boldsymbol{\phi}^{(0)}+C_1^{(4)}\boldsymbol{\phi}^{(1)}+C_2^{(4)}\boldsymbol{\phi}^{(2)}+C_3^{(4)}\boldsymbol{\phi}^{(3)} \quad \cdots\cdots(22)$$

であり,式(18)~(21)を式(22)に代入してディジタル信号系列 $\{x_n\}_{n=0}^{n=3}$ はそれぞれ,

$$\begin{cases} x_0=C_0^{(4)}+1.3065C_1^{(4)}+C_2^{(4)}+0.5411C_3^{(4)} \\ x_1=C_0^{(4)}+0.5411C_1^{(4)}-C_2^{(4)}-1.3065C_3^{(4)} \\ x_2=C_0^{(4)}-0.5411C_1^{(4)}-C_2^{(4)}+1.3065C_3^{(4)} \\ x_3=C_0^{(4)}-1.3065C_1^{(4)}+C_2^{(4)}-0.5411C_3^{(4)} \end{cases} \quad \cdots\cdots(23)$$

となる.したがって,式(17)のDCT値を式(23)のIDCTに代入すれば,

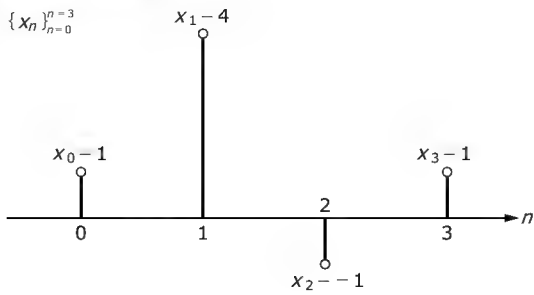
$$x_0=4,\ x_1=4,\ x_2=-6,\ x_3=2 \quad \cdots\cdots(24)$$

となることから,**図7**のディジタル信号系列がDCTの正規直交基底ベクトルに信号分解され,式(23)に基づいて計算することにより,もとの信号波形が再合成される(**図8**).

〔図8〕正規直交基底ベクトルによる信号分解

例題2

図9に示すディジタル信号のDCT値を計算した後,DCT値

〔図9〕 例題2

から再合成できることを検証せよ．

解答2

まず，式(2)よりDCT値を$\left\{C_\ell^{(4)}\right\}_{\ell=0}^{\ell=3}$計算して，以下の結果を得る．

$$\begin{cases} C_0^{(4)}=1.25, & C_1^{(4)}=0.6764, \\ C_2^{(4)}=-0.25, & C_3^{(4)}=-1.6332 \end{cases} \quad \cdots\cdots(25)$$

次に，式(25)のDCT値を式(23)に代入して**図9**のディジタル信号が得られることを確認する(計算省略)．

ディジタルコサイン逆変換(IDCT)の一般式

前述した「DCT値からディジタル信号を再合成する」処理は，ディジタルコサイン変換(DCT)の逆変換に相当し，逆DCT(IDCT ; Inverse DCTの略)とよばれる．IDCTの計算は，式(2)を$\{x_n\}_{n=0}^{n=N-1}$に関するN元連立方程式とみなして解を求めることであり，

$$x_n=\sum_{\ell=0}^{N-1}\gamma_\ell C_\ell^{(N)}\cos\left\{\frac{(2n+1)\ell}{2N}\pi\right\} \quad ; n=0,\ 1,\ 2,\ \ldots,\ (N-1) \quad \cdots\cdots(26)$$

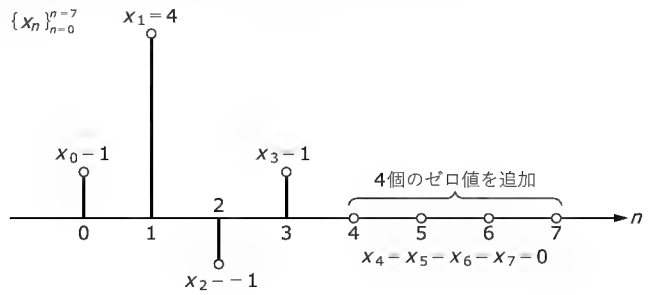
と表される．

たとえば，簡単な例として$N=2$の場合を採り上げてみることにしよう．まずは，式(2)よりDCTは次式で算出される．

$$\begin{cases} C_0^{(2)}=\dfrac{1}{2}\left[x_0\{1\}+x_1\{1\}\right]=\dfrac{1}{2}(x_0+x_1) \\ C_1^{(2)}=\dfrac{1}{2}\left[x_0\left\{\sqrt{2}\cos\left(\dfrac{\pi}{4}\right)\right\}+x_1\left\{\sqrt{2}\cos\left(\dfrac{3\pi}{4}\right)\right\}\right]=\dfrac{1}{2}(x_0-x_1) \end{cases} \quad \cdots\cdots(27)$$

ここで，{ }の中は正規直交基底ベクトルの要素値を表す．

よって，式(27)をディジタル信号のサンプル値(x_0, x_1)について解くことにより，IDCTは，

$$\begin{cases} x_0=C_0^{(2)}+C_1^{(2)} \\ x_1=C_0^{(2)}-C_1^{(2)} \end{cases} \quad \cdots\cdots(28)$$

と表され，式(26)の正規直交基底ベクトルの形式では，

$$\begin{cases} x_0=C_0^{(2)}\{1\}+C_1^{(2)}\left\{\sqrt{2}\cos\left(\dfrac{\pi}{4}\right)\right\}=C_0^{(2)}+C_1^{(2)} \\ x_1=C_0^{(2)}\{1\}+C_1^{(2)}\left\{\sqrt{2}\cos\left(\dfrac{3\pi}{4}\right)\right\}=C_0^{(2)}-C_1^{(2)} \end{cases} \quad \cdots(29)$$

であり，式(28)に一致した結果が見事に得られることになるのである．

〔図10〕ゼロ(0)値を追加したディジタル信号の例

ゼロ(0)値の追加とDCT値の関係

いま，**図9**の4サンプルのディジタル信号に4個のゼロ値を追加し，全部で8サンプルのディジタル信号を考えてみよう(**図10**)．このとき，式(2)に基づき，$N=8$としてDCT値$\left\{C_\ell^{(8)}\right\}_{\ell=0}^{\ell=7}$を計算すると，

$$\begin{cases} C_0^{(8)}=0.625,\ C_1^{(8)}=0.6975,\ C_2^{(8)}=0.3382, \\ C_3^{(8)}=0.0842,\ C_4^{(8)}=-0.125,\ C_5^{(8)}=-0.4828, \\ C_6^{(8)}=-0.8166,\ C_7^{(8)}=-0.6787 \end{cases} \quad \cdots(30)$$

となる(**図11**)．

図11には，DCT値を周波数($0\leqq\omega T<\pi$)に対して連続的に計算した値，すなわち，

$$C^{(N)}(\omega T)=\frac{1}{N}\sum_{n=0}^{N-1}\gamma(\omega T)x_n\cos\left\{\frac{(2n+1)}{2}\omega T\right\} \quad \cdots\cdots(31)$$

$$\text{ただし，}\ \gamma(\omega T)=\begin{cases} 1\ ;\ \omega T=0 \\ \sqrt{2}\ ;\ \omega T\neq 0 \end{cases}$$

を破線で示してある．ここで，この破線で示す特性を“連続的DCT(Continuous DCT，以後，CDCTと略記)”とよぶことにする．一方，DCT値$\left\{C_\ell^{(N)}\right\}_{\ell=0}^{\ell=N-1}$は離散的なので，式(31)のCDCTにおいて周波数が，

$$\omega T=\frac{2\pi}{2N}\ell \quad ; \ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1) \cdots\cdots(32)$$

に対応した値であることも明らかである．つまり，

$$C_\ell^{(N)}=C^{(N)}\left(\frac{2\pi}{2N}\ell\right) \quad \cdots\cdots(33)$$

である．

ところで，**図9**と**図11**のDCT値，すなわち式(25)の $\left\{C_\ell^{(4)}\right\}_{\ell=0}^{\ell=3}$ と式(30)の $\left\{C_\ell^{(8)}\right\}_{\ell=0}^{\ell=7}$ の値を比較してみると，

$$C_{2\ell}^{(8)}=\frac{C_\ell^{(4)}}{2}\ ;\ \ \ell=0,\ 1,\ 2,\ 3 \cdots\cdots\cdots\cdots\cdots\cdots\cdots(34)$$

となる関係が成立することがわかる．式(34)より，サンプル数と同数のゼロ値を追加すると，周波数分解能が，

$$\Delta\omega T^{(4)}=\frac{\pi}{N}$$

から，

$$\Delta\omega T^{(8)}=\frac{\pi}{2N}$$

へと2倍に細かくなり，他方DCT値は半分になるのである．このような性質は，一般的なサンプル数Nに対しても成立し，

$$C_{2\ell}^{(2N)}=\frac{C_\ell^{(N)}}{2}\ ;\ \ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1) \cdots\cdots\cdots(35)$$

と表される(式(2)より明白なので，証明は省略)．なお，連続的なDCT(CDCT)についても式(35)と同様な性質があり，

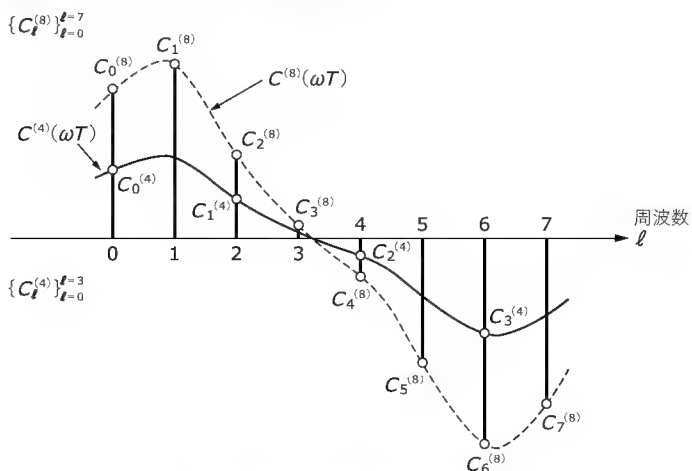
$$C^{(2N)}(\omega T)=\frac{C^{(N)}(\omega T)}{2} \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots(36)$$

となる関係を有する．

例題3

図9のディジタル信号から**図12**のような対称波形を作成したとき，DCT値を求め，もとの信号のDCT値と比較せよ．

解答3

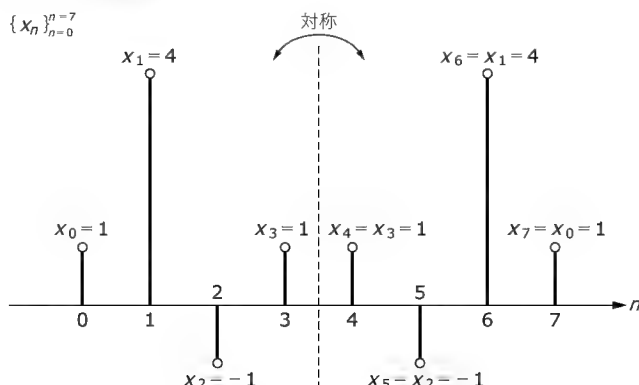
式(2)に**図12**のサンプル値 $\left\{x_n\right\}_{n=0}^{n=7}$ を代入してDCT $\left\{C_\ell^{(8)}\right\}_{\ell=0}^{\ell=7}$ 値を算出する．以下に，計算結果を示す．

$$\begin{cases} C_0^{(8)}=1.25,\ \ C_1^{(8)}=0,\ \ C_2^{(8)}=0.6764 \\ C_3^{(8)}=0,\ \ C_4^{(8)}=-0.25,\ \ C_5^{(8)}=0, \\ C_6^{(8)}=-1.6332,\ \ C_7^{(8)}=0 \end{cases}$$

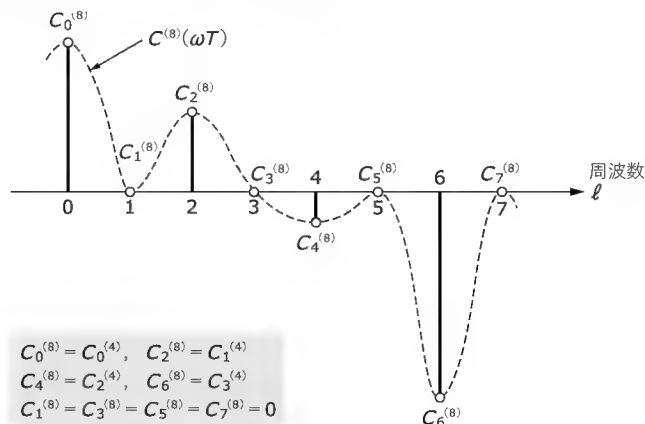
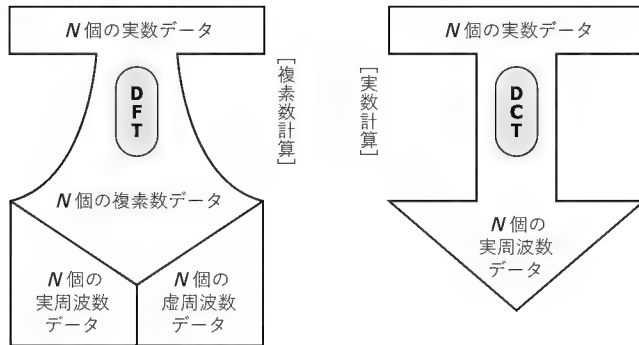
よって，式(25)の $\left\{C_\ell^{(4)}\right\}_{\ell=0}^{\ell=3}$ と見比べてみると，$\left\{C_\ell^{(8)}\right\}_{\ell=0}^{\ell=7}$ の奇数番目の値は0，偶数番目は一つおきに $\left\{C_\ell^{(4)}\right\}_{\ell=0}^{\ell=3}$ に一致することがわかる(**図13**)．一般的には次式が成立するので，式(2)に基づき，各自で検証してもらいたい．

$$\begin{cases} C_{2\ell}^{(2N)}=C_\ell^{(N)} \\ C_{2\ell+1}^{(2N)}=0 \end{cases} ;\ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1) \cdots\cdots\cdots(37)$$

DCTとDFT

DCTは，平たくいえば「N個の実数データをN個の実周波数データに変換するように，DFTを改造したものである」，といえる．つまり，DFTのままでは「N個の実数データをN個の複素数データとして，N個の実周波数データ(実数部)とN個の虚周波数データ(虚数部)に変換してしまう」わけである．

〔図11〕ゼロ値の追加とDCT値の関係

〔図12〕例題3

〔図13〕対称波形とDCT値の関係

〔図14〕DCTとDFT

このDFTのもつ複素数処理を実数計算として実現するために，N個の実数データを対称波形として2倍のサンプル数，すなわち$2N$個の実数データを用意したのである．こうすることによって，DCTでは周波数分解能を2倍に高め，直流近傍の低い周波数成分に重きをおいた効率的なスペクトル分析が可能な手法として，とくにディジタル画像処理分野におけるデータ圧縮技術での不動の地位を確立している(**図14**)．

まとめとして，DCTとDFTとを対比させながら，それぞれの特徴を大ざっぱに記しておく．

[DCT値]

DCT係数ともいう．$C_0^{(N)}$は直流(Direct Current)という意味で「DC係数」，それ以外のDCT値$C_\ell^{(N)}$($\ell \neq 0$)は交流(Alternative Current)という意味から「AC係数」とよばれる．

DCT値はDFT値と非常に密接な関係があり，低い周波数から高い周波数へと順番に並んでいる．

[DFT，DCTの性質]

(1) DFTは複素数値を与え，DCTは実数値を与える．

(2) DFTでは振幅(大きさ，絶対値)は“対称(偶対称ともいう)”，位相(偏角)は“反対称(奇対称ともいう)”となる性質があり，サンプリング周波数の1/2に対して複素共役の値を有する．

(3) DCTは対称性をもたない．

(4) DCTはDFTの大きさとは異なるところもあるが，相互に密接な関係を有する．DFT値が周波数成分に相当すると同様に，DCT値も周波数成分としての物理的な意味をもつ．

例題4

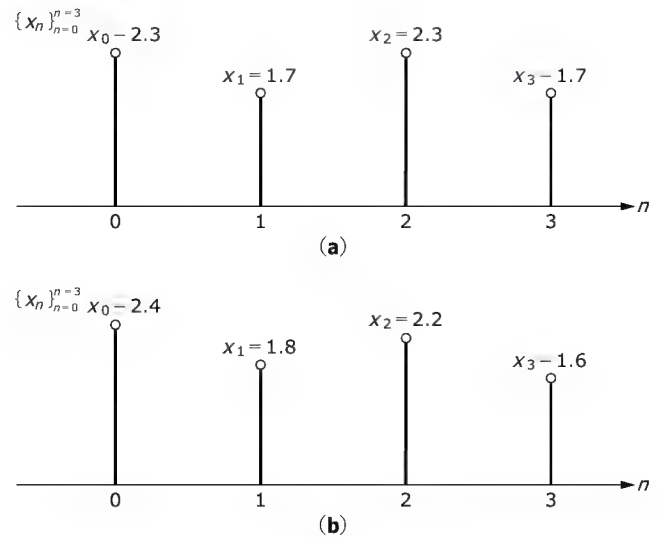
図15(a)，**(b)**に示す4サンプルのディジタル信号について，それぞれDCT値，DFT値を求めよ．

解答4

いずれのディジタル信号も直流近傍の低い周波数成分を多く含んでおり，画像信号などが典型的な例である．それぞれ**表1**，**表2**に，DCTとDFTの計算結果を示しておくので各自で検証してもらいたい．結果からわかることでもあるが，直流成分は同じ値をとることを確認しておいてほしい〔式(6)〕．

〔図15〕例題4

(a)

(b)

〔表1〕図15(a)のディジタル信号のDCTとDFT

ℓ	DCT	DFT
0	2	2
1	0.1148	0
2	0	0.3
3	0.2771	0

〔表2〕図15(b)のディジタル信号のDCTとDFT

ℓ	DCT	DFT
0	2	2
1	0.2071	$0.05-j0.05$
2	0	0.3
3	0.2388	$0.05+j0.05$

表1，**表2**より，直流近傍の周波数が非常に大きい反面，高い周波数成分の割合が小さいといった特徴的な性質を有するデータ(たとえば，画像信号など)に対しては，データ量を圧縮する基本技術として有効であることが類推される．

*　　　　*

次回からは2次元データの画像信号をとくに強く意識しながら，“DCTによる信号解析への応用”を中心に見すえて，わかりやすく解説していく予定である．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科

第11回

GCC2.95から追加変更のあったオプションの補足と検証

岸　哲夫

本連載も10回をすぎた．連載開始当初は一般的だったGCC2.95も少数派になっていることだろう．新旧のバージョンを比較すると，機能が大きく変化しているため，本連載で扱うバージョンをGCC3.3に改めることにする(ただし，今回のみGCC3.3の環境構築が間に合わなかったので，検証はGCC3.2.2で行う)．そこで，今回から2回の予定で，GCC2.95から追加変更のあったオプションの補足と検証を行うことにする．

(筆者)

今後，本連載で扱う予定の項目を以下に記します．

- GCC2.95から追加変更のあったオプションの補足と検証
- GCC2.95から追加変更のあった，その他言語仕様の補足と検証
- GDBを使用するためのデバッグオプションについての検証
- GCCが使用している標準ライブラリの使用法と検証
- GCCのプログラミング
- GTK，GNOME，KDEなどを使ったGUIプログラミング

その後，C++言語やProlog，Javaなども含めたプログラミング言語やスクリプト言語，またデータベースといった多種にわたるGNUツールに関して，解説および検証を行っていく予定です．

さて，GCC3.2.2になり，オプションの種類と使用法が変わってきました．それを2回の予定で説明・検証していきます．

● C言語の方言を扱うオプション

▶-ansi

ANSI規格に沿ったCプログラムをサポートします．つまり，ISO C89規格を採用しています．ここで，C99規格やGCCの拡張機能を使用するとエラーになります(**リスト1**)．

コンパイルした結果を以下に示します．

```
$ gcc -ansi -pedantic test141.c -o test141
$ test141.c:5:25: 警告: 無名可変引数マクロは
                              C99 で採り入れられました
```

この場合，ワーニングエラーにはなりますが，実行は可能です．アセンブラソース(**リスト2**)も正しく展開されています．

以下に実行結果を示します．

```
$ ./test141
debug:x01=100
debug:(100,200)
debug:(100,200,12行目)
$
```

問題なく動作しました．また，デフォルトでコンパイルしても同じ実行結果になりました．

```
$ gcc test141.c -o test141
$ ./test141
debug:x01=100
debug:(100,200)
debug:(100,200,12行目)
$
```

▶-std=

● 指定するパラメータの値

```
c89
iso9899:1990
```

上の値を＝の後に指定するとISO C89規格でコンパイルされます．現在は-ansiを指定したことと同義になります．

先に使用したソース(**リスト1**)をコンパイルすると，結果は以下のようになります．

```
$ gcc -std=c89 -pedantic test141.c
                                   -o test141
test141.c:5:25: 警告: 無名可変引数マクロは
                           C99 で採り入れられました
$ gcc -std=iso9899:1990
           -pedantic test141.c -o test141
test141.c:5:25: 警告: 無名可変引数マクロは
                           C99 で採り入れられました
$
```

● 指定するパラメータの値

```
c99
```

〔リスト1〕**C99規格でエラーになる例**(test141.c)

```
/*
 * 可変個数の引数を持つマクロの例.(C99 規格)
 */
#include <stdio.h>
#define M_debug(format, ...)     printf("debug:" format, __VA_ARGS__)
main()
{
    int  x01  =    100;
    int  x02  =    200;
    M_debug("x01=%d¥n",x01);
    M_debug("(%d,%d)¥n",x01,x02);
    M_debug("(%d,%d,%d行目)¥n",x01,x02,__LINE__ );
}
```

〔**リスト2**〕**生成されたアセンブラソース**（test141_1.s）

```
    .file     "test141.c"
    .section  .rodata
.LC0:
    .string   "debug:x01=%d¥n"
.LC1:
    .string   "debug:(%d,%d)¥n"
.LC2:
    .string   "debug:(%d,%d,%d¥271¥324¥314¥334)¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $100, -4(%ebp)
    movl      $200, -8(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $4, %esp
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC1
    call      printf
    addl      $16, %esp
    pushl     $12
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC2
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔**リスト3**〕**C99規格で生成されたアセンブラソース**（test141_2.s）

```
    .file     "test141.c"
    .section  .rodata
.LC0:
    .string   "debug:x01=%d¥n"
.LC1:
    .string   "debug:(%d,%d)¥n"
.LC2:
    .string   "debug:(%d,%d,%d¥271¥324¥314¥334)¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $100, -4(%ebp)
    movl      $200, -8(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $4, %esp
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC1
    call      printf
    addl      $16, %esp
    pushl     $12
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC2
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

```
c9x
iso9899:1999
iso9899:199x
```

上の値を＝の後に指定すると，ISO C99規格でコンパイルされます．前述したとおり完全にサポートしているわけではありません．

リスト1のソースをコンパイルすると，結果は以下のようになります．

```
$ gcc -std=c99  test141.c -o test141
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ gcc -std=c9x  test141.c -o test141
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ gcc -std=iso9899:1999  test141.c
                                      -o test141
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ gcc -std=iso9899:199x  test141.c
                                      -o test141
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$
```

test141.cでは戻り値を明示的に指定していませんが，C99規格では許されません．そこでワーニングエラーを出します．

リスト3のアセンブラソースを見るとわかるように，デフォルトの戻り値を0として生成しています．

●指定するパラメータの値

```
gnu89
```

上の値を＝の後に指定すると，ISO C89規格とGCC拡張仕様の組み合わせ，かつC99規格を含む言語仕様となります．これが現バージョンのデフォルトです．

リスト1のソースをコンパイルすると，結果は次のようになります．

```
$ gcc -std=gnu89  test141.c  -S
$ cp test141.s test141_3.s
$
```

リスト4のアセンブラソース，およびコンパイル結果を見ると-ansiでもC99規格でもないことがわかります．これがGCC独自の言語仕様です．

●指定するパラメータの値

gnu99

gnu9x

上の値を＝の後に指定すると，ISO C99規格とGCC拡張仕様の組み合わせでコンパイルされます．将来は，これがデフォルトになります．

リスト1のソースをコンパイルすると，結果は以下のようになります．

```
$ gcc -std=gnu99  test141.c  -S
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ gcc -std=gnu9x  test141.c  -S
test141.c:7: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ cp test141.s test141_4.s
$
```

test141.cでは戻り値を明示的に指定していませんが，C99規格では許されません．そこでワーニングエラーを出します．

リスト5のアセンブラソースを見るとわかるように，デフォルトの戻り値を0として生成しています．つまり，C99規格を使用してコンパイルしています．

では，**リスト1**のソースにGCC拡張機能を含む処理を追加して(**リスト6**)，コンパイルしてみます．

```
$ gcc -std=gnu9x  test142.c  -o test142
test142.c:10: 警告: 戻り値の型をデフォルトの
                                    `int' とします
$ gcc -std=gnu9x  test142.c  -S
test142.c:10: 警告: 戻り値の型をデフォルトの
```

〔リスト4〕デフォルトの言語仕様で生成されたアセンブラソース(test141_3.s)

```
    .file     "test141.c"
    .section  .rodata
.LC0:
    .string   "debug:x01=%d¥n"
.LC1:
    .string   "debug:(%d,%d)¥n"
.LC2:
    .string   "debug:(%d,%d,%d¥271¥324¥314¥334)¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $100, -4(%ebp)
    movl      $200, -8(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $4, %esp
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC1
    call      printf
    addl      $16, %esp
    pushl     $12
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC2
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト5〕C99規格で生成されたアセンブラソース(test141_4.s)

```
    .file     "test141.c"
    .section  .rodata
.LC0:
    .string   "debug:x01=%d¥n"
.LC1:
    .string   "debug:(%d,%d)¥n"
.LC2:
    .string   "debug:(%d,%d,%d¥271¥324¥314¥334)¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $100, -4(%ebp)
    movl      $200, -8(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $4, %esp
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC1
    call      printf
    addl      $16, %esp
    pushl     $12
    pushl     -8(%ebp)
    pushl     -4(%ebp)
    pushl     $.LC2
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト6〕C99規格およびGCC拡張機能を含む例(test142.c)

```
/*
 * 可変個数の引数を持つマクロの例(C99規格)
 *typeof でマクロを作る例(GCC拡張規格)
 */
#include <stdio.h>
#define M_debug(format, ...)printf("debug:" format, __VA_ARGS__)
#define pointer(T)  typeof(T *)
#define array(T, N) typeof(T [N])
main()
{
    int ix;
    int  x01  =    100;
    int  x02  =    200;
/*  */
    array (pointer (char), 10) char_p;
    array (pointer (long), 10) long_p;
/*  */
    M_debug("x01=%d¥n",x01);
    M_debug("(%d,%d)¥n",x01,x02);
    M_debug("(%d,%d,%d行目)¥n",x01,x02,__LINE__ );
/*  */
    for (ix=0;ix<10;ix++)
    {
         char_p[ix] = (char *)'a'+ix;
         long_p[ix] = (long *)(ix * 1000000L);
    }
    for (ix=0;ix<10;ix++)
    {
         printf("char_p[%d] = %d¥n",ix,char_p[ix]);
    }
    for (ix=0;ix<10;ix++)
    {
         printf("long_p=[%d] = %d¥n",ix,long_p[ix]);
    }
}
```

〔リスト7〕生成されたアセンブラソース(test142.s)

```
    .file     "test142.c"
    .section  .rodata
.LC0:
    .string   "debug:x01=%d¥n"
.LC1:
    .string   "debug:(%d,%d)¥n"
.LC2:
    .string   "debug:(%d,%d,%d¥271¥324¥314¥334)¥n"
.LC3:
    .string   "char_p[%d] = %d¥n"
.LC4:
    .string   "long_p=[%d] = %d¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $120, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $100, -16(%ebp)
    movl      $200, -20(%ebp)
    subl      $8, %esp
    pushl     -16(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $4, %esp
    pushl     -20(%ebp)
    pushl     -16(%ebp)
    pushl     $.LC1
    call      printf
    addl      $16, %esp
    pushl     $20
    pushl     -20(%ebp)
    pushl     -16(%ebp)
    pushl     $.LC2
    call      printf
    addl      $16, %esp
    movl      $0, -12(%ebp)
.L2:
    cmpl      $9, -12(%ebp)
    jle  .L5
    jmp  .L3
.L5:
    movl      -12(%ebp), %edx
    movl      -12(%ebp), %eax
    addl      $97, %eax
    movl      %eax, -72(%ebp,%edx,4)
    movl      -12(%ebp), %ecx
    movl      -12(%ebp), %edx
    movl      %edx, %eax
    sall      $2, %eax
    addl      %edx, %eax
    leal      0(,%eax,4), %edx
    addl      %edx, %eax
    leal      0(,%eax,4), %edx
    addl      %edx, %eax
    leal      0(,%eax,4), %edx
    addl      %edx, %eax
    leal      0(,%eax,4), %edx
    addl      %edx, %eax
    leal      0(,%eax,4), %edx
    addl      %edx, %eax
    sall      $6, %eax
    movl      %eax, -120(%ebp,%ecx,4)
    leal      -12(%ebp), %eax
    incl      (%eax)
    jmp  .L2
.L3:
    movl      $0, -12(%ebp)
.L6:
    cmpl      $9, -12(%ebp)
    jle  .L9
    jmp  .L7
.L9:
    subl      $4, %esp
    movl      -12(%ebp), %eax
    pushl     -72(%ebp,%eax,4)
    pushl     -12(%ebp)
    pushl     $.LC3
    call      printf
    addl      $16, %esp
    leal      -12(%ebp), %eax
    incl      (%eax)
    jmp  .L6
.L7:
    movl      $0, -12(%ebp)
.L10:
    cmpl      $9, -12(%ebp)
    jle  .L13
    jmp  .L11
.L13:
    subl      $4, %esp
    movl      -12(%ebp), %eax
    pushl     -120(%ebp,%eax,4)
    pushl     -12(%ebp)
    pushl     $.LC4
    call      printf
    addl      $16, %esp
    leal      -12(%ebp), %eax
    incl      (%eax)
    jmp  .L10
.L11:
    movl      $0, %eax
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト8〕生成されたリスト(test142.txt)

```
/* compiled from: . */
/* /usr/include/libio.h:402:NC */ extern int __underflow (_IO_FILE *);
/* /usr/include/libio.h:403:NC */ extern int __uflow (_IO_FILE *);
/* /usr/include/libio.h:404:NC */ extern int __overflow (_IO_FILE *, int);
/* /usr/include/libio.h:405:NC */ extern wint_t __wunderflow (_IO_FILE *);
/* /usr/include/libio.h:406:NC */ extern wint_t __wuflow (_IO_FILE *);
/* /usr/include/libio.h:407:NC */ extern wint_t __woverflow (_IO_FILE *, wint_t);
/* /usr/include/libio.h:432:NC */ extern int _IO_getc (_IO_FILE *);
/* /usr/include/libio.h:433:NC */ extern int _IO_putc (int, _IO_FILE *);
/* /usr/include/libio.h:434:NC */ extern int _IO_feof (_IO_FILE *);
/* /usr/include/libio.h:435:NC */ extern int _IO_ferror (_IO_FILE *);
/* /usr/include/libio.h:437:NC */ extern int _IO_peekc_locked (_IO_FILE *);
/* /usr/include/libio.h:443:NC */ extern void _IO_flockfile (_IO_FILE *);
/* /usr/include/libio.h:444:NC */ extern void _IO_funlockfile (_IO_FILE *);
/* /usr/include/libio.h:445:NC */ extern int _IO_ftrylockfile (_IO_FILE *);
/* /usr/include/libio.h:463:NC */ extern int _IO_vfscanf (_IO_FILE *, const char *, __gnuc_va_list, int *);
/* /usr/include/libio.h:465:NC */ extern int _IO_vfprintf (_IO_FILE *, const char *, __gnuc_va_list);
/* /usr/include/libio.h:466:NC */ extern __ssize_t _IO_padn (_IO_FILE *, int, __ssize_t);
/* /usr/include/libio.h:467:NC */ extern size_t _IO_sgetn (_IO_FILE *, void *, size_t);
/* /usr/include/libio.h:469:NC */ extern __off64_t _IO_seekoff (_IO_FILE *, __off64_t, int, int);
/* /usr/include/libio.h:470:NC */ extern __off64_t _IO_seekpos (_IO_FILE *, __off64_t, int);
/* /usr/include/libio.h:472:NC */ extern void _IO_free_backup_area (_IO_FILE *);
/* /usr/include/stdio.h:154:NC */ extern int remove (const char *);
/* /usr/include/stdio.h:156:NC */ extern int rename (const char *, const char *);
/* /usr/include/stdio.h:163:NC */ extern FILE *tmpfile (void);
/* /usr/include/stdio.h:173:NC */ extern char *tmpnam (char *);
/* /usr/include/stdio.h:183:NC */ extern char *tmpnam_r (char *);
/* /usr/include/stdio.h:196:NC */ extern char *tempnam (const char *, const char *);
/* /usr/include/stdio.h:202:NC */ extern int fclose (FILE *);
/* /usr/include/stdio.h:204:NC */ extern int fflush (FILE *);
/* /usr/include/stdio.h:209:NC */ extern int fflush_unlocked (FILE *);
/* /usr/include/stdio.h:222:NC */ extern FILE *fopen (const char *, const char *);
/* /usr/include/stdio.h:226:NC */ extern FILE *freopen (const char *, const char *, FILE *);
/* /usr/include/stdio.h:252:NC */ extern FILE *fdopen (int, const char *);
/* /usr/include/stdio.h:276:NC */ extern void setbuf (FILE *, char *);
/* /usr/include/stdio.h:281:NC */ extern int setvbuf (FILE *, char *, int, size_t);
/* /usr/include/stdio.h:288:NC */ extern void setbuffer (FILE *, char *, size_t);
/* /usr/include/stdio.h:291:NC */ extern void setlinebuf (FILE *);
/* /usr/include/stdio.h:298:NC */ extern int fprintf (FILE *, const char *, ...);
/* /usr/include/stdio.h:300:NC */ extern int printf (const char *, ...);
/* /usr/include/stdio.h:303:NC */ extern int sprintf (char *, const char *, ...);
/* /usr/include/stdio.h:307:NC */ extern int vfprintf (FILE *, const char *, __gnuc_va_list);
/* /usr/include/stdio.h:310:NC */ extern int vprintf (const char *, __gnuc_va_list);
/* /usr/include/stdio.h:313:NC */ extern int vsprintf (char *, const char *, __gnuc_va_list);
/* /usr/include/stdio.h:321:NC */ extern int snprintf (char *, size_t, const char *, ...);
/* /usr/include/stdio.h:325:NC */ extern int vsnprintf (char *, size_t, const char *, __gnuc_va_list);
/* /usr/include/stdio.h:354:NC */ extern int fscanf (FILE *, const char *, ...);
/* /usr/include/stdio.h:356:NC */ extern int scanf (const char *, ...);
/* /usr/include/stdio.h:359:NC */ extern int sscanf (const char *, const char *, ...);
/* /usr/include/stdio.h:383:NC */ extern int fgetc (FILE *);
/* /usr/include/stdio.h:384:NC */ extern int getc (FILE *);
/* /usr/include/stdio.h:387:NC */ extern int getchar (void);
/* /usr/include/stdio.h:396:NC */ extern int getc_unlocked (FILE *);
/* /usr/include/stdio.h:397:NC */ extern int getchar_unlocked (void);
/* /usr/include/stdio.h:402:NC */ extern int fgetc_unlocked (FILE *);
/* /usr/include/stdio.h:408:NC */ extern int fputc (int, FILE *);
/* /usr/include/stdio.h:409:NC */ extern int putc (int, FILE *);
/* /usr/include/stdio.h:412:NC */ extern int putchar (int);
/* /usr/include/stdio.h:421:NC */ extern int fputc_unlocked (int, FILE *);
/* /usr/include/stdio.h:426:NC */ extern int putc_unlocked (int, FILE *);
/* /usr/include/stdio.h:427:NC */ extern int putchar_unlocked (int);
/* /usr/include/stdio.h:433:NC */ extern int getw (FILE *);
/* /usr/include/stdio.h:436:NC */ extern int putw (int, FILE *);
/* /usr/include/stdio.h:443:NC */ extern char *fgets (char *, int, FILE *);
/* /usr/include/stdio.h:447:NC */ extern char *gets (char *);
/* /usr/include/stdio.h:480:NC */ extern int fputs (const char *, FILE *);
/* /usr/include/stdio.h:483:NC */ extern int puts (const char *);
/* /usr/include/stdio.h:487:NC */ extern int ungetc (int, FILE *);
/* /usr/include/stdio.h:492:NC */ extern size_t fread (void *, size_t, size_t, FILE *);
/* /usr/include/stdio.h:495:NC */ extern size_t fwrite (const void *, size_t, size_t, FILE *);
/* /usr/include/stdio.h:507:NC */ extern size_t fread_unlocked (void *, size_t, size_t, FILE *);
/* /usr/include/stdio.h:509:NC */ extern size_t fwrite_unlocked (const void *, size_t, size_t, FILE *);
/* /usr/include/stdio.h:515:NC */ extern int fseek (FILE *, long int, int);
/* /usr/include/stdio.h:517:NC */ extern long int ftell (FILE *);
/* /usr/include/stdio.h:519:NC */ extern void rewind (FILE *);
/* /usr/include/stdio.h:550:NC */ extern int fgetpos (FILE *, fpos_t *);
/* /usr/include/stdio.h:552:NC */ extern int fsetpos (FILE *, const fpos_t *);
/* /usr/include/stdio.h:577:NC */ extern void clearerr (FILE *);
/* /usr/include/stdio.h:579:NC */ extern int feof (FILE *);
/* /usr/include/stdio.h:581:NC */ extern int ferror (FILE *);
/* /usr/include/stdio.h:586:NC */ extern void clearerr_unlocked (FILE *);
/* /usr/include/stdio.h:587:NC */ extern int feof_unlocked (FILE *);
```

〔**リスト8**〕**生成されたリスト**(test142.txt)(つづき)

```
/* /usr/include/stdio.h:588:NC */ extern int ferror_unlocked (FILE *);
/* /usr/include/stdio.h:594:NC */ extern void perror (const char *);
/* /usr/include/stdio.h:606:NC */ extern int fileno (FILE *);
/* /usr/include/stdio.h:611:NC */ extern int fileno_unlocked (FILE *);
/* /usr/include/stdio.h:618:NC */ extern FILE *popen (const char *, const char *);
/* /usr/include/stdio.h:621:NC */ extern int pclose (FILE *);
/* /usr/include/stdio.h:627:NC */ extern char *ctermid (char *);
/* /usr/include/stdio.h:655:NC */ extern void flockfile (FILE *);
/* /usr/include/stdio.h:659:NC */ extern int ftrylockfile (FILE *);
/* /usr/include/stdio.h:662:NC */ extern void funlockfile (FILE *);
/* test142.c:10:OF */ extern int main (void); /* () */
```

〔**リスト9**〕**組み込み関数を含む例**(test143.c)

```
/*
 *組み込み関数について
 */
#include <stdlib.h>
#include <stdio.h>
main()
{
    int  a;
    a    =    abs(-5);
    printf("%d¥n",a);
}
```

`int' とします

$

両方の言語仕様を含むソースでも，問題なくコンパイルできます．では，実行してみましょう．

```
$ ./test142
debug:x01=100
debug:(100,200)
debug:(100,200,20行目)
char_p[0] = 97
char_p[1] = 98
char_p[2] = 99
char_p[3] = 100
char_p[4] = 101
char_p[5] = 102
char_p[6] = 103
char_p[7] = 104
char_p[8] = 105
char_p[9] = 106
long_p=[0] = 0
long_p=[1] = 1000000
long_p=[2] = 2000000
long_p=[3] = 3000000
long_p=[4] = 4000000
long_p=[5] = 5000000
long_p=[6] = 6000000
long_p=[7] = 7000000
long_p=[8] = 8000000
long_p=[9] = 9000000
```

結果とアセンブラソース(**リスト7**，p.175)を見ればわかるように，どちらの機能も生きています．

▶-aux-info filename

ヘッダファイル中のものを含むすべてのプロトタイプ(**リスト8**)を出力します．

```
$ gcc  test142.c -aux-info test142.txt
```

▶-fno-builtin，-fno-builtin-function

ライブラリ関数のうち，abort，abs，alloca，cos，exit，fabs，ffs，labs，memcmp，memcpy，sin，sqrt，strcmp，strcpy，strlenは効率を良くするために，組み込み関数としてコンパイルされることがあります．

その場合，デバッガを使用する際に意図しないふるまいをしたり，ライブラリ関数の処理自体が意図しないふるまいをすることがあります．それを防止するために「組み込みにしない」設定ができます．-fno-builtinは組み込み関数を，いっさい使用しません．

なお，以下に示すtest143.c(**リスト9**)とtest144.cは名前が違う同一のソースです．

```
$ gcc  -fno-builtin test143.c -S
$ gcc  test144.c -S
```

アセンブラソースを見ると，test143.s(**リスト10**)では，call　absとなっていますが，test144.s(**リスト11**)では組み込み関数を呼び出しているようです．

〔**リスト10**〕**test143.cから生成されたアセンブラソース**(test143.s)

```
    .file     "test143.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    subl      $12, %esp
    pushl     $-5
    call      abs
    addl      $16, %esp
    movl      %eax, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト11〕test144.c から生成されたアセンブラソース（test144.s）

```
    .file     "test144.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $5, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト12〕test145.c から生成されたアセンブラソース（test145.s）

```
    .file     "test145.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $5, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト13〕test146.c から生成されたアセンブラソース（test146.s）

```
    .file     "test146.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $5, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

〔リスト14〕test143.c から生成されたシンボルリスト（test143nm.txt）

```
080493ec D _DYNAMIC
080494c8 D _GLOBAL_OFFSET_TABLE_
080483d4 R _IO_stdin_used
         w _Jv_RegisterClasses
080494b8 d __CTOR_END__
080494b4 d __CTOR_LIST__
080494c0 d __DTOR_END__
080494bc d __DTOR_LIST__
080493e8 d __EH_FRAME_BEGIN__
080493e8 d __FRAME_END__
080494c4 d __JCR_END__
080494c4 d __JCR_LIST__
080494e4 A __bss_start
080493dc D __data_start
08048390 t __do_global_ctors_aux
080482f0 t __do_global_dtors_aux
080493e0 d __dso_handle
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
080494e4 A _edata
080494e8 A _end
080483b4 T _fini
080483d0 R _fp_hw
08048250 T _init
080482a8 T _start
         U abs@@GLIBC_2.0
080482cc t call_gmon_start
080494e4 b completed.1
080493dc W data_start
0804832c t frame_dummy
08048358 T main
080493e4 d p.0
         U printf@@GLIBC_2.0
```

〔リスト15〕test144.c から生成されたシンボルリスト（test144nm.txt）

```
080493b0 D _DYNAMIC
0804948c D _GLOBAL_OFFSET_TABLE_
08048398 R _IO_stdin_used
         w _Jv_RegisterClasses
0804947c d __CTOR_END__
08049478 d __CTOR_LIST__
08049484 d __DTOR_END__
08049480 d __DTOR_LIST__
080493ac d __EH_FRAME_BEGIN__
080493ac d __FRAME_END__
08049488 d __JCR_END__
08049488 d __JCR_LIST__
080494a4 A __bss_start
080493a0 D __data_start
08048354 t __do_global_ctors_aux
080482c0 t __do_global_dtors_aux
080493a4 d __dso_handle
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
080494a4 A _edata
080494a8 A _end
08048378 T _fini
08048394 R _fp_hw
08048230 T _init
08048278 T _start
0804829c t call_gmon_start
080494a4 b completed.1
080493a0 W data_start
080482fc t frame_dummy
08048328 T main
080493a8 d p.0
         U printf@@GLIBC_2.0
```

なお，-fno-builtin-functionというオプションもありますが，現バージョンでは何もしないことになっています．

以下に示すtest145.cとtest146.cは名前が違う同一のソースで，test143.cと同じです．

```
$ gcc  -fno-builtin-function test145.c -S
$ gcc  -fbuiltin-function test146.c -S
```

リスト12および**リスト13**を見ればわかるように，どちらを指定しても組み込み関数を呼び出しています．

それは，**リスト14**，**リスト15**に示すシンボルリストでも確認できます．test143の場合，関数abs()は標準ライブラリ内のものをリンクしています．

▶-no-integrated-cpp

GCCのサブプログラムには，cpp，cc1，as，ldがあります．そして，-Bオプションの指定でサブプログラムがどこにあるかを指定できます．つまり，指定の方法によっては標準でないcppを内部で実行することが可能になります．このオプションを指定すると，その機能を無視して標準のcppを使用します．なお，このオプションは将来も存続するかどうか不明だそうです．

標準でないcppを使う場合には，インストール時のカスタマイズで指定したほうが混乱しないでしょう．

● LINK関連のオプション

▶-shared-libgcc，-static-libgcc

シェアードライブラリとしてlibgccを提供するシステムにおいては，このオプションによってライブラリを共有するかスタティックにするかを決定します．

ただし，コンパイラが構築されたとき，libgccの共有が指定されていなければ，これらのオプションは効果がありません．

ちなみに，Red Hat 8.0環境のGCC3.2.2では以下のとおり，共有されていません．

```
$ gcc -v
/usr/lib/gcc-lib/i386-redhat-linux/3.2/
                     specs から spec を読み込み中
コンフィグオプション: ../configure --prefix=
        /usr --mandir=/usr/share/man --infod
ir=/usr/share/info --enable-shared --enable
          -threads=posix --disable-checking
            --host=i386-redhat-linux --with
         -system-zlib --enable-__cxa_atexit
スレッドモデル: posix
gcc バージョン 3.2 20020903
                    (Red Hat Linux 8.0 3.2-7)
$
```

● メッセージ関連のオプション

▶-fmessage-length=n

このオプションは，表示されるエラーメッセージの長さをフォーマットするものです．デフォルトは72です．しかし，日本語の場合には意図したとおりにならないようです．次のようにずれます．

```
$ gcc -std=gnu9x  test142.c
                         -fmessage-length=20

test142.c:10: 警告: 戻り値の型をデフォルトの
                              `int' とします
$ gcc -std=gnu9x  test142.c
                         -fmessage-length=50
test142.c:10: 警告: 戻り値の型をデフォルトの
                              `int' とします
$ gcc -std=gnu9x  test142.c
                         -fmessage-length=60
test142.c:10: 警告: 戻り値の型をデフォルトの
                              `int' とします
$ gcc -std=gnu9x  test142.c
                         -fmessage-length=55
test142.c:10: 警告: 戻り値の型をデフォルトの
                              `int' とします
$
```

▶-fdiagnostics-show-location=every-line

このオプションを指定すると，上の例でいうと“test142.c:10: 警告: ”が各行に表示されます．

```
$ gcc -std=gnu9x  test142.c
                 -fdiagnostics-show-location=
every-line -fmessage-length=55
test142.c:10: 警告: 戻り値の型をデフォルトの `int'
test142.c:10: 警告: とします
$
```

▶-fdiagnostics-show-location=once

前述の例で，“test142.c:10: 警告: ”が各行に表示されなくなります．これがデフォルトのふるまいです．

```
$ gcc -std=gnu9x  test142.c
                 -fdiagnostics-show-location=
                     once -fmessage-length=55
test142.c:10: 警告: 戻り値の型をデフォルトの
   `int' とします
$
```

● プリプロセッサ関連のオプション

▶-Wcomments

コメントが矛盾している場合にチェックします．たとえば/*が重複した場合などにワーニングエラーとします．

リスト16のソースをコンパイルした結果は，以下のとおりです．

```
$ gcc test145.c  -Wcomments
test145.c:1:3: 警告: コメント内に "/*" があります
$ gcc test145.c
$
```

▶-Wsystem-headers

システムヘッダ中に問題があり，ワーニングエラーとなる場合でも通常はメッセージを表示しません．しかし，このオプションを指定することで，すべてのワーニングを表示します．あまり使わないとは思います．

▶-MF file

以下の-M?オプションは，Makefileを作るときに有用なコマンドです．個々のソースの依存関係を明確にします．

-Mや-MMとともに使用した場合は，指定されたデータファイルに依存情報を書き込みます．

-MDや-MMDとともに使用した場合，本来のデータファイルではなく，指定されたデータファイルに情報を書き込みます．

以下のように指定します(**リスト17**，**リスト18**)．

```
$ gcc -M -MF depdata_M.txt test146.c
$
```

以下のように出力されたデータファイルの内容は，-Mオプションで標準出力に出力されたものと同一です．

```
$ gcc -M  test146.c
test146.o: test146.c test146.h test146a.h
                    test146b.h test146c.h ¥
  test146d.h /usr/include/stdio.h
                  /usr/include/features.h ¥
  /usr/include/sys/cdefs.h
                  /usr/include/gnu/stubs.h ¥
  /usr/lib/gcc-lib/i386-redhat-linux/
                   3.2/include/stddef.h ¥
  /usr/include/bits/types.h
          /usr/include/bits/pthreadtypes.h ¥
  /usr/include/bits/sched.h /usr/include/
          libio.h /usr/include/_G_config.h ¥
  /usr/include/wchar.h /usr/include/
        bits/wchar.h /usr/include/gconv.h ¥
  /usr/lib/gcc-lib/i386-redhat-linux/3.2/
                        include/stdarg.h ¥
  /usr/include/bits/stdio_lim.h /usr/
                include/bits/sys_errlist.h
$
```

-MMとともに使用する場合，以下のように指定します．

```
$ gcc -MM -MF depdata_MM.txt test146.c
```

▶-MP

このオプションは分割コンパイルするすべてのソースの依存情報を表示します．-MMや-Mとともに使います．以下では，-MMとともに使った標準出力をdepdata_MP.txt(**リスト23**)に書き込んでいます．それぞれのソースを**リスト19～リスト22**に示します．

```
$ gcc -MM -MP  test146.c  test147.c
```

〔リスト16〕コメントが矛盾している場合の例(test145.c)

```
/*/*
 *組み込み関数について
 */
#include <stdlib.h>
#include <stdio.h>
main()
{
    int  a;
    a    =    abs(-5);
    printf("%d¥n",a);
}
```

〔リスト17〕依存関係を表示する例

```
/*
 *依存関係
 */
#include "test146.h"
int main()
{
    a    =    1;
    pr01();
    pr02();
    pr03();
    return    0;
}
```

(**a**) test146.c

```
#include "test146a.h"
void     pr01();
void     pr02();
void     pr03();
int a;
```

(**b**) test146.h

```
//依存関係の試験
#include "test146b.h"
```

(**c**) test146a.h

```
//依存関係の試験
#include "test146c.h"
```

(**d**) test146b.h

```
//依存関係の試験
#include "test146d.h"
```

(**e**) test146c.h

```
//依存関係の試験
#include <stdio.h>
```

(**f**) test146d.h

〔リスト18〕依存関係を出力したリスト(depdata_M.txt)

```
test146.o: test146.c test146.h test146a.h test146b.h test146c.h ¥
  test146d.h /usr/include/stdio.h /usr/include/features.h ¥
  /usr/include/sys/cdefs.h /usr/include/gnu/stubs.h ¥
  /usr/lib/gcc-lib/i386-redhat-linux/3.2/include/stddef.h ¥
  /usr/include/bits/types.h /usr/include/bits/pthreadtypes.h ¥
  /usr/include/bits/sched.h /usr/include/libio.h /usr/include/_G_config.h ¥
  /usr/include/wchar.h /usr/include/bits/wchar.h /usr/include/gconv.h ¥
  /usr/lib/gcc-lib/i386-redhat-linux/3.2/include/stdarg.h ¥
  /usr/include/bits/stdio_lim.h /usr/include/bits/sys_errlist.h
```

〔リスト19〕依存関係を表示する例(test147.c)

```
extern   int  a;
int b;
void     pr01()
{
    b    =    1;
    a    =    10;
    printf("pr01");
}
```

〔リスト20〕依存関係を表示する例(test148.c)

```
void     pr02()
{
    printf("pr02t146.h");
}
```

〔リスト21〕依存関係を表示する例(test149.c)

```
extern   int  b;
#include "test146.h"
void     pr03()
{
    b    =    0;
    printf("pr03t146.h");
    pr04();

}
```

〔リスト22〕依存関係を表示する例(test150.c)

```
#include "test146.h"
void     pr04()
{
    printf("pr04t146.h");
    printf("pr01t146.h");
    a    =    11;

}
```

test148.c test149.c test150.c　> depdata_MP.txt

どれがどのソースやヘッダをincludeしているのか，混乱した場合には，これで明確になるでしょう．

▶-fpreprocessed

あまり使用しないオプションだと思いますが，説明します．前処理(preprocessed)されたソースをコンパイルしたいときに使います．

通常のソースで指定すると，以下のような矛盾が出ます．

```
$ gcc   test146.c  test147.c test148.c
        test149.c test150.c  -fpreprocessed

test146.c:4: '#' トークンの所で文法エラー
test146.c:4: 文字列定数の前に 構文解析エラー
test146.c:8: 警告: データ定義が型や記憶クラスを
                                    持っていません
test146.c:9: 警告: データ定義が型や記憶クラスを
                                    持っていません
test146.c:10: 警告: データ定義が型や記憶クラスを
                                    持っていません
test146.c:11: 構文解析エラー が "return" の
                                    前にあります
test149.c:2: '#' トークンの所で文法エラー
test149.c:2: 文字列定数の前に 構文解析エラー
test149.c:6: 文字列定数の前に 構文解析エラー
test149.c:6: 警告: 組み込み関数 `printf' と
                                    型が矛盾します
test149.c:6: 警告: データ定義が型や記憶クラスを
                                    持っていません
test149.c:7: 警告: データ定義が型や記憶クラスを
                                    持っていません
test149.c:9: 構文解析エラー が '}' トークンの
                                    前にあります
test150.c:1: '#' トークンの所で文法エラー
test150.c:1: 文字列定数の前に 構文解析エラー
test150.c:5: 文字列定数の前に 構文解析エラー
test150.c:5: 警告: 組み込み関数 `printf' と
                                    型が矛盾します
test150.c:5: 警告: データ定義が型や記憶クラスを
                                    持っていません
test150.c:6: 警告: データ定義が型や記憶クラスを
                                    持っていません
test150.c:8: 構文解析エラー が '}' トークンの
                                    前にあります
$
```

-Eオプションを指定すると，前処理したソースを標準出力に出力します(連載第4回参照)．

そこで，前処理したソース(**リスト24**)を-fpreprocessedオプション付きでコンパイルしてみます．

```
$ gcc   -E test142.c  > prepro.c
$ gcc -fpreprocessed prepro.c -o test142
$ ./test142test142.c  > prepro.c test142
debug:x01=100
debug:(100,200)
debug:(100,200,20行目)
char_p[0] = 97
char_p[1] = 98
char_p[2] = 99
char_p[3] = 100
char_p[4] = 101
char_p[5] = 102
char_p[6] = 103
char_p[7] = 104
char_p[8] = 105
char_p[9] = 106
long_p=[0] = 0
long_p=[1] = 1000000
long_p=[2] = 2000000
long_p=[3] = 3000000
long_p=[4] = 4000000
```

〔**リスト23**〕**依存関係を出力したリスト**(depdata_MP.txt)

```
test146.o: test146.c test146.h test146a.h test146b.h test146c.h ¥
  test146d.h

test146.h:

test146a.h:

test146b.h:

test146c.h:

test146d.h:
test147.o: test147.c
test148.o: test148.c
test149.o: test149.c test146.h test146a.h test146b.h test146c.h ¥
  test146d.h

test146.h:

test146a.h:

test146b.h:

test146c.h:

test146d.h:
test150.o: test150.c test146.h test146a.h test146b.h test146c.h ¥
  test146d.h

test146.h:

test146a.h:

test146b.h:

test146c.h:

test146d.h:
```

〔リスト 24〕前処理したソースリスト(途中省略)(prepro.c)

```
# 1 "test142.c"
# 1 "<built-in>"
# 1 "<¥245¥263¥245¥336¥245¥363¥245¥311¥245¥351¥245¥244¥245¥363>"
# 1 "test142.c"



# 1 "/usr/include/stdio.h" 1 3
# 28 "/usr/include/stdio.h" 3
# 1 "/usr/include/features.h" 1 3
# 291 "/usr/include/features.h" 3
# 1 "/usr/include/sys/cdefs.h" 1 3
# 292 "/usr/include/features.h" 2 3
# 320 "/usr/include/features.h" 3
# 1 "/usr/include/gnu/stubs.h" 1 3
# 321 "/usr/include/features.h" 2 3
# 29 "/usr/include/stdio.h" 2 3



         省略

extern char *ctermid (char *__s) ;
# 655 "/usr/include/stdio.h" 3
extern void flockfile (FILE *__stream) ;



extern int ftrylockfile (FILE *__stream) ;


extern void funlockfile (FILE *__stream) ;
# 679 "/usr/include/stdio.h" 3

# 6 "test142.c" 2
```

```


main()
{
        int ix;
        int x01 = 100;
        int x02 = 200;

        typeof(typeof(char *) [10]) char_p;
        typeof(typeof(long *) [10]) long_p;

        printf("debug:" "x01=%d¥n", x01);
        printf("debug:" "(%d,%d)¥n", x01,x02);
        printf("debug:" "(%d,%d,%d行目)¥n", x01,x02,20);

        for (ix=0;ix<10;ix++)
        {
                char_p[ix] = (char *)'a'+ix;
                long_p[ix] = (long *)(ix * 1000000L);
        }
        for (ix=0;ix<10;ix++)
        {
                printf("char_p[%d] = %d¥n",ix,char_p[ix]);
        }
        for (ix=0;ix<10;ix++)
        {
                printf("long_p=[%d] = %d¥n",ix,long_p[ix]);
        }
}
```

〔リスト 25〕エラーになるソースリスト(test151.c)

```
/*
 * エラーになるソース
 */
int main()
    {
        int  a          =    1;
        return    0;
    }
```

```
long_p=[5] = 5000000
long_p=[6] = 6000000
long_p=[7] = 7000000
long_p=[8] = 8000000
long_p=[9] = 9000000
$
```

このように，正常に動作しました．

CPU パワーが低く，プリプロセスに時間がかかるため，プリプロセスを省略して修正し，コンパイルしたいときに使うと有効です．

▶-ftabstop=width

これは，たとえばコンパイラが出力するメッセージにカラム数まで表示されているときに，このソースのタブ幅は何桁かを指定し，きっちりカラム数を表示させるために使います．もっとも，現状ではあまり使わないような気がします．

リスト 25 はソースの最後に改行を入れていませんが，このときエラーにはなるものの，10 カラム目だということを正確に表示しています．

```
$ gcc test151.c -ftabstop=8
test151.c:8:10: 警告: ファイル末尾に改行がありません
$
```

▶-fno-show-column

このオプションを指定すると，コンパイラが出力するメッセージにカラム数を表示しません．

先のソースで実行すると以下のようになります．

```
$ gcc test151.c -fno-show-column
test151.c:8: 警告: ファイル末尾に改行がありません
$
```

▶-no-gcc

既存のソースで，GCC の場合に特別な処理をしているとして，もしそれが不要な場合，GNU C ではないものとしてコンパイルする方法があります．-no-gcc オプションを付けると，__GNUC__，__GNUC_MINOR__，__GNUC_PATCHLEVEL__ は defined ではないとみなします．

リスト 26 のコンパイルおよび実行結果を以下に示します．

```
$ gcc test152.c -no-gcc -o test152
$ ./test152
GNU  Cのソースではありません
$ gcc test152.c -o test152
$ ./test152
```

〔リスト26〕**GNU Cではないとみなしてコンパイルする例**(test152.c)

```
/*
 *GNU  Cではないとみなしてコンパイルする
 */
int main()
    {
#if defined __GNUC__
    printf("GNU  Cのソースです¥n");
#else
    printf("GNU  Cのソースではありません¥n");
#endif
         int  a         =    1;
         return    0;
    }
```

GNU　Cのソースです
$

▶**-remap**

MS-DOSのように非常に短いファイル名しか許されないファイルシステム環境で特別なコードを実行させることができます．

▶**-$**

通常のGNU Cでは，$の使用を禁じていません．もし，禁じる必要がある際には，このオプションを指定します．リテラル中の$は無視します．

リスト27のコンパイル結果は以下のようになります．

```
$ gcc test153.c
$ gcc test153.c  -$
test153.c: 関数 `main' 内:
test153.c:11: プログラムとして逸脱した文字 '$'
$
```

● 警告を要求/抑止するオプション

次に，ワーニングを出すか出さないかを制御するオプションについて説明します．

▶**-Wformat**

このオプションを指定すると，通常は間違っていてもワーニングを出さない，`printf`系や`scanf`系の関数のフォーマット部分のエラーをチェックします．

リスト28のコンパイルと実行の結果は以下のようになります．

```
$ gcc test154.c
$ gcc test154.c -Wformat
test154.c: 関数 `main' 内:
test154.c:7: 警告: フォーマットは double ですが,
                    引数は different type です (引数 2)
$
```

▶**-Wno-format-y2k**

`-Wformat`オプションを指定したとき，`strftime`を使って2桁の年号を取り出そうとすると，`-Wformat`の機能でワーニングを出します．実際に2桁の年が欲しいのに大きなお世話なのですが，このオプションを指定することで，そのワーニングを出さないようにします．

〔リスト27〕**$を使用禁止した例**(test153.c)

```
/*
 *$の使用禁止
 */
int main()
    {
#if defined __GNUC__
    printf("GNU  Cのソースです$¥n");
#else
    printf("GNU  Cのソースではありません$¥n");
#endif
         int  $a        =    1;
         return    0;
    }
```

〔リスト28〕**printfなどのフォーマットをチェックする例**(test154.c)

```
/*
 *printfなどのフォーマットをチェックする
 */
int main()
{
    printf("%d¥n",9);
    printf("%a¥n",9);
    return    0;
}
```

〔リスト29〕**strftimeのフォーマット中の%gなどをチェックしない例**(test155.c)

```
/*
 *フォーマットをチェックするが
 *strftimeのフォーマットの中で
 *2桁の年を指定してもエラーにしない
 */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main()
{
    char time_str[255];
    struct    tm   *lt;
    time_t    t;
    time(&t);
    lt=localtime(&t);
    strftime(time_str, strlen(time_str), "%g",lt);
    printf("%s¥n",time_str);
    return    0;
}
```

リスト29のコンパイルと実行の結果は以下のようになります．

```
$ gcc test155.c -o test155
```

フォーマットのチェックをしなければ何もメッセージを出しません．

```
$ gcc test155.c -o test155 -Wformat
test155.c: 関数 `main' 内:
test155.c:17: 警告: `%g' は年の下二桁だけを
                                もたらします
```

ここで，`-Wformat`を指定するとワーニングメッセージを出します．

```
$ gcc test155.c -o test155 -Wformat
                              -Wno-format-y2k
```

加えて，`-Wno-format-y2k`を指定するとワーニングメッ

セージを抑止します．

```
$ ./test155
03
$
```

▶-Wno-format-extra-args

-Wformatオプションを指定したときに，printf系やscanf系の関数において引き数が超過していたときのワーニングメッセージを抑止します．

デバッグ中でもない限り，この状態を放置しておくのは混乱の元です．最終的にはこのオプションは取ってコンパイルすべきです．

リスト30のコンパイルと実行の結果は以下のようになります．

```
$ gcc test156.c -o test156
$ gcc test156.c -o test156 -Wformat
test156.c: 関数 `main' 内:
test156.c:12: 警告: フォーマットへの引数が多すぎます
$ gcc test156.c -o test156 -Wformat
                          -Wno-format-extra-args
$ ./test156
0
$
```

▶-Wformat-nonliteral

printf系やscanf系の引き数のフォーマット文字列が，リテラルでない場合にワーニングメッセージを出します．-Wformatオプションを指定したときに有効です．

該当の引き数が本当に意図したものになっているかどうかを確認するのには有効です．実際にフォーマット文字列引き数を文字列変数にした場合は，プログラマがチェックする以外に方法がありません．

リスト31のコンパイルと実行の結果は以下のようになります．

```
$ gcc test157.c -o test157
$ gcc test157.c -o test157 -Wformat
$ gcc test157.c -o test157 -Wformat
                          -Wformat-nonliteral
test157.c: 関数 `main' 内:
test157.c:12: 警告: フォーマットは文字列リテラル
        ではありませんので，引数の型は検査されません
$
```

▶-Wformat-security

関数の戻り値をprintf系関数に指定することはできますが，これがsprintfでメモリ上の転送先が重要な領域で，その関数がシェルだった場合などには，何をしこまれるかわかりません．

非常に簡単なチェックですが，printfなどの引き数が関数の戻り値である時にワーニングメッセージを出します．これも-Wformatオプションとともに使用します．

リスト32のコンパイルと実行の結果は以下のようになります．

```
$ gcc test158.c -o test158
$ gcc test158.c -o test158 -Wformat
$ gcc test158.c -o test158 -Wformat
                          -Wformat-security
test158.c: 関数 `main' 内:
test158.c:15: 警告: フォーマットは非文字列リテラルで,
                  且つフォーマット引数を持ちません
$ ./test158
-Wformat-securityの検証
23
test for -Wformat-security
test
5
$
```

▶-Wformat=2

現在のバージョンにおいて，このオプションは-Wformatオプションに-Wformat-securityオプションと-Wformat-nonliteralオプションの意味を追加したものです．

前述のソースtest158.c（**リスト32**）を使ってコンパイルすると以下のようになります．

```
$ gcc test158.c -o test158 -Wformat=2
test158.c: 関数 `main' 内:
test158.c:13: 警告: フォーマットは文字列リテラルでは
        ありませんので，引数の型は検査されません
test158.c:15: 警告: フォーマットは非文字列リテラルで,
```

〔リスト30〕printfなどの引き数の超過をチェックしない例(test156.c)

```
/*
 *フォーマットをチェックするが
 *printfなどの引数が超過していても
 *エラーにしない
 */
#include <stdio.h>

int main()
{
    int  arg1 =    0;
    int  arg2 =    1;
    printf("%d¥n",arg1,arg2);
    return   0;
}
```

〔リスト31〕フォーマット文が文字列リテラルではない例(test157.c)

```
/*
 *フォーマットが文字列リテラルではない例
 */
#include <stdio.h>

int main()
{
    char *data;
    int  arg1 =    0;
    int  arg2 =    1;
    data =    "%d";
    printf(data,arg1,arg2);
    return   0;
}
```

〔**リスト 32**〕**printf などの引き数が関数の戻り値である例**(test158.c)

```
/*
 *printf などの引き数が関数の戻り値である例
 */
#include <stdio.h>
char    *    testfunc();
char    *    testfunc_format();
char    *    testfunc_arg();
int main()
{
    int  *test;
    char *data;
    data =    "%s%n¥n";
    printf(data,"-Wformat-security の検証",test);
    printf("%d¥n",*test);
    printf(testfunc_arg());
    printf(testfunc_format(),testfunc(),test);
    printf("%d¥n",*test);
    return    0;
}
char    *    testfunc()
{
    return    "test¥n";
}
char    *    testfunc_format()
{
    return    "%s%n";
}
char    *    testfunc_arg()
{
    return    "test for -Wformat-security¥n";
}
```

```
                              且つフォーマット引数を持ちません
test158.c:16: 警告: フォーマットは文字列リテラル
                    ではありませんので，引数の型は検査されません
$
```

▶-Wmissing-braces

連載第7回で配列の初期化について説明しましたが，配列の初期化方法に関してワーニングメッセージを出すオプションです．

```
int tbl1[2][2] = { 0, 1, 2, 3 };
```

上の方法で多次元配列を初期化することは可能ですが，可読性に欠けます．

```
int tbl1[4] = { 0, 1, 2, 3 };
```

これは，上のコードとは意味合いが同じでも二つを混同すると問題が起きます．そこで，下記のようにプログラマが多次元配列だと認識して初期化するべきです．

```
int tbl1[2][2] = { { 0, 1 }, { 2, 3 } };
```

リスト 33 のコンパイルと実行の結果は以下のようになります．

```
$ gcc test159.c -Wmissing-braces
test159.c: 関数 `main' 内:
test159.c:7: 警告: 初期化子のまわりのブレースを
                                        欠いています
test159.c:7: 警告: (`tbl1[0]' の初期化は不完全
                                              です)

$ ./test159
tbl1[0][0]=0
tbl1[0][1]=1
```

〔**リスト 33**〕**配列の初期化について**(test159.c)

```
/*
 * 配列の初期化
 */
#include <stdio.h>
int main(void)
{
    int tbl1[2][2] =    { 0, 1, 2, 3 };
    int tbl2[2][2] =    { { 0, 1 }, { 2, 3 } };
    int tbl3[2][2] =    { [0 ... 1][0 ... 1]=5};
    printf("tbl1[0][0]=%d¥n",tbl1[0][0]);
    printf("tbl1[0][1]=%d¥n",tbl1[0][1]);
    printf("tbl1[1][0]=%d¥n",tbl1[1][0]);
    printf("tbl1[1][1]=%d¥n¥n",tbl1[1][1]);
    printf("tbl2[0][0]=%d¥n",tbl2[0][0]);
    printf("tbl2[0][1]=%d¥n",tbl2[0][1]);
    printf("tbl2[1][0]=%d¥n",tbl2[1][0]);
    printf("tbl2[1][1]=%d¥n¥n",tbl2[1][1]);
    printf("tbl3[0][0]=%d¥n",tbl3[0][0]);
    printf("tbl3[0][1]=%d¥n",tbl3[0][1]);
    printf("tbl3[1][0]=%d¥n",tbl3[1][0]);
    printf("tbl3[1][1]=%d¥n¥n",tbl3[1][1]);
    return;
}
```

```
tbl1[1][0]=2
tbl1[1][1]=3

tbl2[0][0]=0
tbl2[0][1]=1
tbl2[1][0]=2
tbl2[1][1]=3

tbl3[0][0]=5
tbl3[0][1]=5
tbl3[1][0]=5
tbl3[1][1]=5

$
```

▶-Wsequence-point

標準のC言語仕様で規定されていない計算の順序などで，問題が起こりそうなコードを見つけたらワーニングメッセージを出力します．

リスト 34 のコンパイルの結果は，以下のようになります．

```
$ gcc test160.c
$ gcc test160.c -Wsequence-point
test160.c: 関数 `main' 内:
test160.c:9: 警告: `a' での演算が定義されて
                                いないと思われます
$
```

現在のマシン環境において，このような方法で速度を稼いだり，コードを小さくする方法は避けるべきではないかと思います．

生成されたアセンブラのリスト(**リスト 35**)を見ると，a = a+=1, (a++ && n++), a+=1, a+=10, n+=50；の行は左から右に演算しているようです．

▶-Wunused-function

スタティックとして宣言した関数が使われなかったり，定義されていなかったときにワーニングメッセージを出します．

リスト36のコンパイル結果は以下のようになります．

```
$ gcc test161.c
$ gcc test161.c -Wunused-function
test161.c:5: 警告: `test_func1' が `static'
                              と宣言されましたが未定義です
test161.c:18: 警告: `test_func2' が定義されま
                              したが使われませんでした
$
```

▶-Wunused-label

このオプションを付けると，プログラム中で使用していないラベルがある場合にワーニングメッセージを出力します．

それを抑止するには，GCC3.2.2の新機能であるunused attributeの指定をすることです．それに関しては「GCC2.95から追加変更のあったその他言語仕様の補足と検証」の回で解説する予定です．

リスト37のコンパイル結果は以下のようになります．

```
$ gcc test162.c
$ gcc test162.c -Wunused-label
test162.c: 関数 `main' 内:
test162.c:8: 警告: ラベル `label1' が
                        定義されましたが使われていません
$
```

▶-Wunused-parameter

このオプションを付けると，宣言した関数の引き数をプログラム中で使用していない場合にワーニングメッセージを出力します．

それを抑止するには，先と同様にGCC3.2.2の新機能であるunused attributeの指定をすることです．それに関しては，「GCC2.95から追加変更のあったその他言語仕様の補足と検証」の回で解説する予定です．

リスト38のコンパイル結果は，以下のようになります．

```
$ gcc test163.c
$ gcc test163.c -Wunused-parameter
test163.c: 関数 `test_func2' 内:
test163.c:12: 警告: 引数 `arg1' が未使用です
test163.c:12: 警告: 引数 `arg2' が未使用です
$
```

〔リスト34〕演算の順序について(test160.c)

```
/*
 *演算の順序について
 */
#include <stdio.h>
int main(void)
{
    int  a    =    100;
    int  n    =    200;
    a    =    a+=1,(a++ && n++),a+=1,a+=10,n+=50;
    return;
}
```

〔リスト36〕スタティック関数の宣言と実体の矛盾例(test161.c)

```
/*
 *スタティック関数の宣言と実体の矛盾
 */
#include <stdio.h>
static   void test_func1();
static   void test_func2();
void     test_func3();
int main(void)
{
    printf("main¥n");
    return    0;
}
/*static void test_func1()
{
    printf("test_func1¥n");
}*/
static   void test_func2()
{
    printf("test_func2¥n");
}
```

〔リスト37〕使っていないラベル(test162.c)

```
/*
 *使っていないラベル
 */
#include <stdio.h>
int main(void)
{
    printf("main¥n");
label1:
    return    0;
}
```

〔リスト35〕生成されたアセンブラソース(test160.s)

```
    .file     "test160.c"
    .text
    .align 2
.globl main
    .type     main,@function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $100, -4(%ebp)
    movl $200, -8(%ebp)
    leal -4(%ebp), %eax
    incl (%eax)
    movl -4(%ebp), %eax
    movl %eax, -4(%ebp)
    leal -4(%ebp), %eax
    incl (%eax)
    cmpl $1, -4(%ebp)
    je   .L2
    leal -8(%ebp), %eax
    incl (%eax)
.L2:
    leal -4(%ebp), %eax
    incl (%eax)
    leal -4(%ebp), %eax
    addl $10, (%eax)
    leal -8(%ebp), %eax
    addl $50, (%eax)
    leave
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7)"
```

▶-Wunused-variable

このオプションを付けると，定義したスタティック変数やローカル変数をプログラム中で使用していない場合にワーニングメッセージを出力します．

それを抑止するには，先と同様にGCC3.3の新機能であるunused attributeの指定をすることです．それに関しては「GCC2.95から追加変更のあったその他言語仕様の補足と検証」の回で解説する予定です．

リスト39のコンパイル結果は以下のようになります．

```
$ gcc test164.c
$ gcc test164.c -Wunused-variable
test164.c: 関数 `main' 内:
test164.c:10: 警告: 変数 `b' は使われませんでした
test164.c: トップレベル:
test164.c:7: 警告: `a' が定義されましたが
                                    使われませんでした
$
```

＊　＊

次回は，GCC2.95から追加変更のあったオプションの補足と検証の続きを解説する予定です．

きし・てつお

〔リスト38〕使っていない引き数(test163.c)

```
/*
 * 使用していない関数の引数
 */
#include <stdio.h>
void     test_func2(int arg1,int arg2);
void     test_func3();
int main(void)
{
    printf("main¥n");
    return    0;
}
void     test_func2(int arg1,int arg2)
{
    printf("test_func2¥n");
}
```

〔リスト39〕使っていない変数(test164.c)

```
/*
 * 使用していないローカル変数とスタティック変数
 */
#include <stdio.h>
void     test_func2(int arg1,int arg2);
void     test_func3();
static   int  a;
int main(void)
{
    long b;
    printf("main¥n");
    return    0;
}
void     test_func2(int arg1,int arg2)
{
    printf("test_func2¥n");
}
```

● 連合艦隊と無線通信

世界最強といわれたバルチック艦隊を一気に壊滅させた「日本海海戦」は，日本海軍の名を世界にとどろかせた歴史上の有名な史実である．

時は明治38年5月27日午前10時ごろ，哨戒にあたっていた「信濃丸」が，ロシアのバルチック艦隊を彼方に発見し，「敵艦見ゆ」と電文を打った．旗艦「三笠」[注]でこの電文を受けた連合艦隊総司令官，東郷平八郎大将（のち元帥）は「連合艦隊は直ちに出動，敵を撃滅せんとす．本日天気晴朗なれども波高し」という有名な電文を大本営に発信するのと同時に，全軍の士気を鼓舞し，一気に戦闘体制に突入した．

好天のなか，午後1時半ごろ，日本艦隊は敵の大艦隊をはるか遠方に確認した．当然，ロシア艦隊も日本艦隊を確認できただろう．それから30分ほどして，東郷は全艦隊を敵前にて150度左に大反転させる．後に伝説となる「トーゴーターン」だ．

ここでちょっと説明しなければならない．戦闘時には，砲撃艦は一列や山の形に並ぶのが常識だった．これは自軍に自らの砲撃が当たらないようにするためだ．しかし，そのときのロシア艦隊は，長い遠征のため2列縦隊だった．日本艦隊を発見してわずか30分では艦隊を一列に整え直す暇もないだろうが，しかしこのまま攻撃したのでは十分に力を発揮できない．目の前の日本艦隊が反転したのは，まさにそんな時だ．これを見たロシア艦隊は好機到来とばかりに，攻撃を開始した．いったん攻撃開始した以上，砲弾の飛び交う中で艦隊を組み直すことは難しい．

決死の日本艦隊は，しばらくの間そんなロシア艦隊の砲撃に耐えた．晴天で波が高かったため，遠方からの不正確な砲撃となり命中率が下がったことが幸いし，日本艦隊に被害はほとんどなかった．そして，まっすぐ進んできたロシア艦隊の先頭が日本の正確な射程距離に入ったとき，そこには横を向いて一列に並んだ日本艦隊があった．この両軍の体制がT字戦略といわれる所以だ．日本艦隊は砲撃艦全艦で，ロシア艦隊の先頭めがけて集中砲火を浴びせた．ロシア艦はあっという間に沈没した．

大口径砲をもつ戦艦は日本4隻に対しロシア11隻と，戦力的には日本が圧倒的に不利にもかかわらず，戦いが始まって30分も経ったところで，日本の勝利は決していた．世界海戦史上，これほど大規模な戦いはほかにないし，またこれほど決定的な差がついた戦いもないといわれる．

T字戦略は当時の東郷の機転によるものとされていたが，じつはそうではないらしい．この戦略は，日本海軍と大本営が練りに練り，何度も訓練をして好機を伺っていたものだ．先の電文で「天気晴朗なれども波高し」は，じつは作戦成功の大きなカギを伝えた電文だったのである．

日本海海戦の勝因はT字戦略のみではない．無線で敵の到来を早く知り戦闘体制を十分に整えることができたからであることを見逃していけない．

マルコーニが大西洋横断の無線通信に世界で初めて成功したのは1901年である．日本海海戦の明治38年は1904年だから，日本海軍はマルコーニの発明からわずか3年で，世界最先端の無線機を主要艦に積んでいたことになる．

● 巨大艦隊NTT

新聞などで，NTT連合艦隊とか母艦とかいう表現を頻繁に目にする．売り上げが日本の国家予算の8分の1を超えるその巨大さや，強いグループの結束力などを称して，良い意味でも悪い意味でも艦隊と呼んでいるのだろう．

そんなNTTグループの決算が5月13日に発表された．売上高は前期比0.9％減の10兆9231億円となり，1952年の電電公社発足以来，グループで初の減収となった．一方，10万人規模のリストラなどで営業利益は1兆3635億円と急回復したため，体面上大きな問題ではないようにも思える．

NTTが現在の形になったのは1999年7月だが，その分割プランは1996年に決定したものだ．そのとき，現在のようなインターネットやブロードバンドの普及，ユビキタス環境やIP電話の登場といった，パラダイムシフトを予想できたはずはない．当然ながら，現在のNTT連合艦隊は十分戦うことができない編成だという懸念がある．その点で利益は確保したとしても，初の減収はインパクトが大きい．

たとえばYahoo!BBは，2003年5月段階で300万ユーザーを獲得し，多くの競合企業が追従しようとキャンペーンを繰り広げている．一方でNTTが社命をかけて取り組んでいる光ファ

注：旗艦三笠は，世界三台記念艦の一つとして，横須賀の三笠公園に永久保存されている．

イバ事業は，全国でまだ20万世帯にすぎない．

多くの問題をはらみつつも，IP電話が確実に広がるだろう．NTT東西の固定電話は減り続けるだろうが，現在までのようにNTTドコモの契約数増加で補うことはできない．

1996年以来，NTTの再々編を何度も何度も話題にしていたのは公正取引委員会だった．その公取が総務省に統合されてNTT管轄の元郵政省と一緒になってからは，残念というか予想どおりというか，再々編の話は話題にのぼらなくなった．今回はNTTグループの経営陣が自ら危機感を訴え，NTT再々編を口にし始めているようだ．今度こそ自分たちの考えで再分割し，グループとしての競争力を高めたいという意思だろう．

● 顧客サービスの改善

マイラインフィーバーは2001年11月1日で終わり，以来登録変更すると800円の手数料がかかる．しかしNTT以外では登録変更しても，ADSLなどの料金が毎月200円ほど安くなったり，1,000円の商品券をくれるサービスも多い．そんな連絡をくれたりするのも，親切そうなお譲さん（に思える人）だったりする．

最近，NTTから連絡が入った．「シャベリッチの使用料が無料だったのですが，あなたはマイラインプラスに登録されていないことがわかりました．マイラインプラスの契約をするか，シャベリッチ使用料を払うか，解約するかをXX日までに決めてください」というものだった．あまりにも唐突で一方的な電話だった．

シャベリッチとは，毎月一定の費用を払えば市外通話が割安になるNTTのサービスのことだ．月のサービス費用はマイラインプラスに登録していると無料になるということで，マイラインプラスの登録キャンペーンでよく使われた．そこで，筆者もNTTにマイラインプラスとシャベリッチを共に申し込んだのだ．

にもかかわらず，当のNTTから，1年半もたって，あなたは片方しか登録していませんでしたといわれてしまったわけで，何が何だかわけがわからなくなった．

気をとりなおして，NTTに確認の電話を入れてみた．すると番号が違うといわれ新しい番号を教わった．こうして5箇所くらい回された．筆者も少し頭にきて文句をいわせてもらったら，「こっちもリストラでたいへんなんです」と言い返されてしまった．

面倒だからもういいとあきらめていたら数日して，今度は

NTTの代理店から電話があった．「マイラインプラスの登録料800円は当社が負担するので，どうか登録し直してください」という話だった．なぜ代理店に連絡が行ったのかわからなかったが，代理店も，いろいろたいへんなんですとぼやいていたのが印象的だった．

いろいろ考え結局，自宅や事務所のマイラインプラス契約をすべて某社に変えてしまった．商品券をくれるという優しそうなお姉さんの声を思い出したからだ．

NTTの再々編もいいが，顧客サービスを置き去りにしないでほしいと思うのは筆者だけだろうか．

連合艦隊とも称されるNTTは，今期初の減収増益で，まさに「天気晴朗なれども波高し」の現実に直面している．最先端の通信技術をつかって，世界一と称されることになった日本海軍は，民衆を置き去りにすることで大きな失敗を起こして結局は解体する．NTTを非難するつもりは毛頭ない．むしろ現実を見つめ，頑張ってほしいと思う．

あさひ・しょうすけ　テクニカルライター
イラスト　森　祐子

Engineering Life in

凄腕女性エンジニアリングマネージャ(第一部)

■今回のゲストのプロフィール
エリン・トゥルーロロス(Erin Turullols)：中国系アメリカ人，南カリフォルニア出身．シリコンバレー地元にある名門大学，スタンフォード大学電子工学部にて学士号(1994年)と修士号(1996年)を取得．電子工学部では，コンピュータアーキテクチャを専門とする．Hewlett Packardの高速ハイエンドチップ設計専門のラボに所属し，チップセットの開発に携わる．その後，マネジメントに興味をもち，プロジェクトマネージャを務める．2001年にPA-RISC開発グループがインテルに譲渡され，それとともにインテルに入社する．現在，サーバ系プロセッサ開発グループのマネージャを務める．趣味はキックボクシング，テニス，バレーボール，サルサダンスなど．

☆ 仕事のチャンスが生まれると考えてエンジニアリングを選ぶ

トニー エリンさんにはいつもジムでお世話[注1]になっていますが，今日は本業のエンジニアリングの話をお願いします．ラストネームがなかなか難しい発音ですよね？

エリン 夫がメキシコ人，スペイン人，チェコ人，アイルランド人のミックスで，彼の苗字なんです．

トニー なるほど．エンジニアリングに入られたきっかけをお話しいただけますか？

エリン まずスタンフォードに入学したころは，何をしたいのか自分自身でもわかりませんでした．しかし，父がもともと電子工学部出身で博士号ももっており，航空工学のロッキードやJPL(Jet Propulsion Laboratory[注2])で仕事をしています．父の話を聞き，エンジニアになればいろいろな仕事に就けると考えたのです．ハイテクの会社だと，財務やマーケティングもすべて元エンジニアがやっていますから．ちなみに弟が二人いますが，1人はPeopleSoftでアプリケーションエンジニアをしていますし，もう1人はUC Berkeleyに在学中で情報工学と音楽を合わせた独自の専門を勉強中です．

トニー ご家族の皆さんともエンジニアリングになじみが深いのですね．シリコンバレーに就職されたきっかけは？

エリン スタンフォード大学にいたころ，地元のシリコンバレーにはさまざまな会社がたくさんあるので，自分のキャリアをスタートするには最適だと考えてエンジニアリングに入りました．スタンフォード大学の電子工学部では，コンピュータ，電子回路，DSPの三つの大きな専門に分かれていて，私はコンピュータを選びました．

トニー そうですよね，大学で何を専攻し何を専門に勉強するかは，まだ社会経験のない若者には難しい判断ですよね．大学時代にインターンなど，企業でバイトをしませんでしたか？

エリン もちろんやりました．まずは，JPLでインターンをしたのですが，これは父のコネが効いたようです(笑)．当初は，技術ドキュメントのライター兼エディターをやりました．その後は，監視/制御システムのGUIの設計をしました．これはソフトウェアのプロジェクトですね．このシステムは，宇宙船や人工衛星の状態のデータをやり取りするシステムでした．次の年は，HP(Hewlett Packard)のコロラド州にあるグループでマーケティングの仕事にチャレンジしました．半導体テスタなどを作っていて，そのマーケティングに関する仕事でした．最後は，シリコンバレーのCupertino市にあるHPのラボで仕事をしました．仕事とそこにいた人達がすごく気に入ったので，卒業後はここに入りました．ちなみに，Sun Microsystemsでも面接をしてパスしていたのですが，HPを選びました．

☆ 管理職を追及する決意をする

トニー かなりいろいろとトライされてから最終的な就職先を決められましたね．それで仕事のほうはどうでした？

エリン HPの開発ラボで，ハイアベイラビリティ(HA)サーバに使うチップセットの設計グループに所属しました．このシステムには，32個のPA-RISCプロセッサが使われ，メモリも100Gバイト以上ありました．ここでもいろいろな仕事をしました．たとえば，メモリボード用チップのロジック設計と検証などです．また，I/Oデバイスのカスタムレイアウトもしました．ハイエンドなサーバに入るデバイスなので，超高速I/Oなど難しいチャレンジがあったのですが，非常にやりがいのあるプロジェクトでした．

トニー かなり大がかりなデバイス設計ですね．さて，その後はマネジメントのほうに行かれたのですか？

エリン 3年ほど実際の設計に携わったのですが，人と接する仕事のほうに魅力を感じたのです．この手の大きなプロジェクトはチームを組みますが，そのリーダーやチームワークを取る仕事に興味をもちました．そこで，やっぱりこれは管理職になるしかないなと思い，社内での面接を受けました．まあ，当時の上司の後押しとかもあったのですけど．通常は，最低6年ぐらいエンジニアをした人でないと管理職になれないそうで，しかも2回ほどトライしてやっとなれるそうです．でも，私の場合は良いポジションがあったため，すぐに昇進できました．プログラムマネージャと呼ばれる管理職です．チップがテープアウトしたあと，これから実際に量産に入る段階の細かい作業をコーディネートしていく仕事です．たとえば，ソフトウェアグループや社外の協力会社に確保するES(エンジニアリングサンプル)をファブの方に頼んだり，量産に入る段階での検証のチェックリストを管理したり……．

トニー なかなか早い出世ですね．プログラムマネージャはアメリカではよく使われる名称ですが，コーディネータ的な役割ですよね．

注1：筆者の通うジムでキックボクシング(エアロビクスのようなクラスで，ボクセサイズとも呼ばれる)を教えている．

注2：NASAや宇宙開発で有名な会社．南カリフォルニアロサンゼルス郊外にある．

Silicon Valley

H. Tony Chin

対談編

エリン　そうですね，何かを管理しているところではマネージャなのですが，私の場合は仕事を委任したり直属のスタッフがいるわけではないので，非常にフラストレーションを感じました(笑)．細かい作業が多く，自分の所属するグループ以外に他のグループや社外の人達と仕事をするのですが，しかも自分の直属のスタッフではないので，ただ「お願い」をするだけだったのです．完全な管理職，マネジメントというよりは，コーディネータに近い仕事でした．

☆ プロジェクトマネージャの仕事

エリン　次に，また違う管理職になるチャンスがあり，そのプロジェクトでは実際にチームを任されました．8名のスタッフがいて，プロジェクトも非常にわかりやすい内容でした．人数も手頃だし，初めての管理職としてはよかったと思います．作業は，同じくサーバ系のデバイス開発グループの一部で，ICファブから上がってきたES[注3]デバイスのテストツールを作ることでした．当時，初めてのIA64ベースのハードウェアツールで，疑似乱数をベースにパターンを生成してデバイスをテストするものです．数億トランジスタになる大きなデバイスでした．

トニー　ポストシリコンの検証ツールの作成ですね．

エリン　ええ．その後，デバイス開発ラボ全体がインテルに譲渡され，それと一緒にグループごとインテルに移籍しました．私の担当していた検証グループとRTL開発グループが一緒になり，グループが17名のエンジニアで膨れ上がりました．

トニー　そうですよね，Verilog言語のRTL[注4]を書く人と検証する人は大体同じですよね．でも人数が2倍以上に増えたわけですね．スタッフの経験レベルなどは？

エリン　新卒にほぼ近いようなエンジニアもいますし，20年以上のベテランもいます．2名のベテランは，技術リーダー(Technical Lead，省略してTL)になってもらっています．細かい技術的な作業やリーダー的役目をやってくれたり，私の技術的な補佐役をしてくれます．TLがいてくれるおかげで，私はあまり細かい技術的な内容まで見なくてよいのですが，逆に私の技術的な知識/経験にはプラスじゃないですよね．現在のプロジェクトでは，TLに大きな機能ブロック，I/O，メモリI/F，プロセッサI/Fなどのそれぞれを任せています．マイクロアーキテクト的な判断やグループ内のRTLコーディングガイドラインを決めたり，コードレビューなどもします．

トニー　なるほど，TLが細かいところを見ていたり情報を上げてくれるわけですね？　それでは，エリンさんがもっとも時間を使うところは？

注3：Engineering Sample
注4：Register Transfer Level

エリン・トゥルーロロス氏

エリン　TL達はデータや意見/アドバイスを述べてくれますが，判断を下したり，決断するのはすべて私の責任になります．大きなプロジェクトなので，スケジュール的な管理がいちばん大切ですね．上流にはシステムシミュレーションのグループ，下流にはレイアウトグループがいるので，自分達が遅れると影響を与えますから．グループ全体の士気を上げて何とか辛い時期を乗り越えたり，引っ張っていく努力が必要です．

ちょうど今はテープアウトする寸前でとても忙しいです．War Room(戦争時に使われる作戦司令室の意味)を設置して，ほかのマネージャ達と毎日3時間ほどミーティングをしています．進捗状況を把握したり問題点の洗い出しなどを行い，皆で知恵を出し合います．エンジニアリングだけやっているとどうしても目先の問題に集中してしまうのですが，私の場合は全体像を見渡して判断を下すことを心がけています．

トニー　TLは技術的な情報を提供してくれるけれど，決断や判断はしないわけですね．そして，やはり管理職はどの国でも会議やミーティングが多いですよね．そのほかの時間は？

エリン　毎日2時間ぐらいは，スタッフのいる場所を歩き回って話を直接聞いたり，問題点を洗い出します．大体何かあるのですよね．たとえば，ツールやワークステーションがダウンしているのに，ITサポートグループがすぐ来てくれないとか…….

トニー　あっ！　インテルで有名なManagement By Walkingですよね?!　現場を歩きながらマネジメントする方法ですね．

エリン　う～ん，そういうのありましたよね．別に意識してやっているのではありませんが……．自分から実際の作業をしているエンジニアのほうに出向いて行ったほうがわかりやすいと思うので，そうしているだけなんです．あとは，メールが毎日100通以上くるので，これにまた最低2時間近く費やしていますね．メールは，ほかのグループとの連絡や会議の議題やフォローをやり取りしたりによく使います．また，私の日課にはプロジェクトのさまざまな数値データやスケジュール的なデータをアップデートします．

トニー　もちろん，メールの数も多いですよね…….　数値目標はバグ検出率とかいろいろありますよね．これで客観的にさまざまな人にどれくらい作業が進んでいるかわかるようにするわけですね．

次回の予告

引き続きエリンさんに話を聞く．管理職の話を聞いたり，プライベートと仕事のバランス，インテルとHPの違いなど，興味深い話がかなり出た．

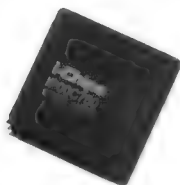
トニー・チン　htchin@attglobal.net　WinHawk Consulting

HARD WARE

●解像度変換 LSI

IP00C750(SCW1)

- 2系統の独立した拡大/縮小エンジンを内蔵し，PiP/PoPや，静止画のタイリングによるマルチ画像表示が可能.
- RGB24ビット/YUV 4:2:2 16ビット，YUV 4:4:4 24ビット，108M画素/秒の2系統の画像入力を装備.
- RGB30ビット，85M画素/秒のW-XGAパネル出力をもつ.
- 縦横独立な倍率設定ができ，4:3画像から16:9画像への変換が容易に実現可能.
- ワイドパネル向けに，垂直/水平方向パノラマ変換機能をサポート.
- 入出力画像ポートが独立に動作し，フレームレート変換や追い越し制御を容易に実現.

■ アイチップス・テクノロジー(株)
価格：下記へ問い合わせ
TEL：06-6492-7277　FAX：06-6492-7388

●A-D コンバータ

ADCDS-1405

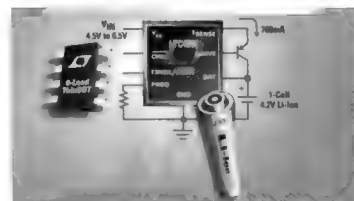
- 14ビット/5MHzの画像信号処理専用サンプリングA-Dコンバータ.
- CCDセンサ特有の残留電荷，チャージインジェクションあるいは熱雑音を取り除くための相関ダブルサンプリング機能を備え，高精度画像処理システムに適する.
- 性能は，±0.9LSB(保証値)の微分非直線性，76dBのSFDR，71dBのSINAD(*S/N*含歪)となっている.
- ±5V，12Vの三電源で動作し，消費電力は900mW.
- TTL/CMOS入出力ロジック互換.
- 使用温度範囲は0～70℃で，－55～125℃のADCDS-1405EXモデルも用意.
- パッケージは，40ピンTDIPで提供.

■ デイテル(株)
価格：¥38,800(1～24個時)
TEL：03-3779-1031　FAX：03-3779-1030

●バッテリチャージャ

LTC4056

- マイクロコントローラから独立したソリューションで，ACアダプタやUSBポートなど，安定していない入力電源や低下している入力電源からの高速充電を可能にする，ThinSOTパッケージのリニアリチウムイオンバッテリチャージャコントローラ.
- 総面積75mm²，高さ1mmであるため，携帯ハンドヘルド機器のスペース要件を満たす.
- 入力電圧源が低下している場合，入力電源が回復するまで充電電流を制限しながら，バッテリ充電を継続することで，充電時間を短縮.
- 充電，充電終了，保護に必要な機能をすべて搭載.

■ リニアテクノロジー(株)
サンプル価格：¥165～(1,000個時)
TEL：03-5226-7291　FAX：03-5226-0268

●LCD ドライバチップセット

HD66781/HD66783

- HD66781は，26万色表示に対応した表示用RAMと表示制御用のコントローラを内蔵した720出力のソースドライバ.
- HD66783は，液晶駆動電圧発生用の電源回路を内蔵した328出力のゲートドライバ.
- QVGAサイズで26万色表示のアモルファスTFTカラー液晶パネルに対応でき，パネルを含む消費電力は，従来品の画面サイズとほぼ同等の5mWの低消費電力を実現.
- 新機能としてOSD(オンスクリーンディスプレイ)機能や透過表示用のアルファブレンディング機能，画像の拡大，縮小用のリサイズ機能などの多様な表示機能を搭載.

■ (株)ルネサステクノロジ
サンプル価格：¥2,200(HD66781)
¥800(HD66783)
TEL：03-5201-5226

●プログラマブルチップ

Virtex-Ⅱ Pro X

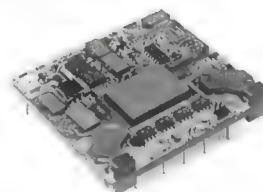
- トランシーバモジュールに組み込まれたハードIPを含んでおり，OC-48 SONET適合システムの構築，OC-192データ伝送速度以上でのSONETの導入をするための光トランシーバの直接ドライブが可能.
- チャネルあたり2.488Gbps～10.3125Gbpsの伝送速度をサポートする，RocketIO XマルチGビットトランシーバを搭載.
- 10Gビット Ethernet，10Gファイバチャネル，SxI-5，TFI-5，PCI Express，Serial RapidIO，XFPおよびVSR光系などの標準伝送システムをサポート.
- 組み込まれたRocketIO Xトランシーバは，オプションで8B/10Bおよび64B/66Bのコーディング，20ビット/16ビットデータバス，20x/16xクロッキング，チャネルボンディング，およびプリエンファシス，受信等化，差動信号生成，およびチップ終端などのMGT性能を最適化するためのプログラマブル機能を備える.

■ ザイリンクス(株)
価格：下記へ問い合わせ
TEL：03-5321-7740　FAX：03-5321-7762

●DC-DC コンバータ

UHPシリーズ

- ハーフブリック オープンフレーム形状を採用し，61×58.5×10.7mmの小型サイズを実現したDC-DCコンバータ.
- 1.5/1.8/2.5/3.3Vの4種類の出力電圧モデルをそろえ，45～60Aの大出力電流容量を実現.
- 定格入力電圧は48Vdcで，入力電圧範囲は36～75V.
- センス，出力電圧調整端子付き.
- オン/オフ制御端子付き.
- 過大出力保護，短絡保護，過昇温度保護付き.
- UL/EN60950承認，CEマーク付き.
- 2250Vdc BASIC絶縁.

■ デイテル(株)
価格：¥13,600(1～9個時)
TEL：03-3779-1031　FAX：03-3779-1030

HARD WARE

●ポテンショメータ

HSM22E 型

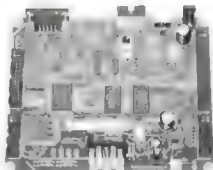
- 独自の回路構成，内部構造により，EMC 耐性に優れる回転無接触ポテンショメータ．
- ホール IC を使用した無接点構造のために寿命が長く，耐振動性に優れている．
- 防衛庁認定規格（MIL など）に相当する環境試験条件をクリア．
- 外形寸法が φ 23mm で，回転寿命が約 1 億回の高信頼性タイプ．
- 従来品とのアタッチメントが共通で，ジョイスティックコントローラ（L シリーズ）への流用が可能．

■ 栄通信工業（株）
サンプル価格：¥3,000
TEL：044-411-5580　FAX：044-434-2520
E-mail：sales@sakae-tsushin.co.jp

●ボードコンピュータ

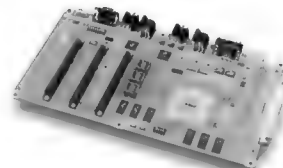
TEC2638U-CAN

- マルチタスク方式の通信プロトコル「CAN」ドライバおよび USB インターフェース搭載の，組み込み用ボードコンピュータ．
- CAN 内蔵 1 チップマイコン（日立製 H8/2638F16 ビット CPU）を使用．
- 全ソースコードおよび CPU 内蔵フラッシュメモリの書き込みソフトウェアが標準添付されているため，コンパイラを用意するだけで製品開発に着手できる．
- 出荷時に書き込まれたテスト用ファームウェアは，フリーの ITRON Ver.4（HOS-V4）を利用して作成されている．
- CAN は，マルチタスク方式のバス構成になっている．

■（有）テクノクラフト
価格：下記へ問い合わせ
TEL：042-793-0256　FAX：042-793-0440
E-mail：techno@tt.rim.or.jp

●NTSC ビデオデコーダ/エンコーダ

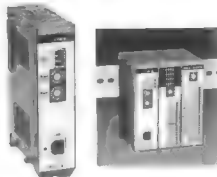
MU200-VD
MU200-XSR

- MU200-VD はディジタルビデオ信号の試作/検証環境を容易に実現する NTSC ビデオデコード/エンコードコンポーネント，MU200-XSR は SRAM メモリボードである．
- MU200-VD ビデオデコード/エンコードコンポーネントは，FPGA コンポーネント MU200-AP シリーズと接続することにより，ディジタルビデオ信号処理の試作，検証環境を実現可能．
- アルテラ社製の最新 FPGA を搭載し，画像処理（画像の合成，圧縮/伸張，認識，効果などの研究開発）に適したコンポーネント．

■ 三菱電機マイコン機器ソフトウエア（株）
価格：下記へ問い合わせ
TEL：075-958-3574　FAX：075-958-3782
E-mail：medusa@kyo.mms.co.jp
URL：http://www.mms.co.jp/

●USB 搭載 I/O コントローラモジュール

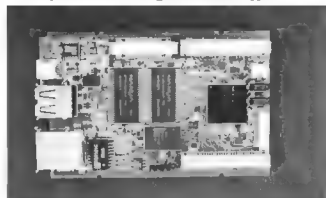
CPU-CA10(USB)GY

- パソコンと USB ケーブルで接続することにより，パソコンで集中制御を行うリモート I/O システムの構築が可能．
- 入出力インターフェース拡張用製品「デバイスモジュール」シリーズのディジタル入出力，アナログ入出力，カウンタ入力を最大 8 台まで側面にスタックすることが可能．
- 市販の USB ハブを利用することで，1 台のパソコンに最大 127 台を接続することが可能．
- パソコンのセットアップが手早く簡単に行える「プラグ&プレイ機能」，電源を投入したままで抜き差しが行える「ホットプラグ機能」を装備．
- 省電力，低発熱 CPU を採用し，ファンレスを実現．

■（株）コンテック
価格：¥38,000
TEL：03-5628-9286　FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●Linux 搭載 CPU ボード

M-CARD

- V_R4120CPU をコアに，SDRAM コントローラ，CF card，SIO，A-D/D-A，LCD，SB，ISA バスなどの各種 I/O 機能を内蔵し，消費電力は 200mW．
- SDRAM は，64M ～ 256M ビットを 2 個搭載可能で，最大 64M バイトで 32 ビット接続．
- フラッシュメモリは，64M ～ 128M ビットを 1 個搭載可能で，最大 16M バイトで 16 ビット接続．
- LAN は，AX88796L（ASIX）10/100Base-TX モジュラジャック搭載．
- USB はホスト 1 チャネル，ファンクション 1 チャネルを搭載し，いずれも Rev1.1 準拠．

■ メガソリューション（株）
価格：下記へ問い合わせ
TEL：03-3874-1557　FAX：03-5603-2314
E-mail：info@megasolution.jp
URL：http://www.megasolution.jp/

●ミッドレンジ IP アクセスルータ

GeoStream Si-R500

- IPsec と QoS 機能の同時利用により，安価なインターネット VPN においても，遅延やゆらぎのない高品質かつリアルタイムな通信を実現し，IP 電話サービスを高い通話品質で利用可能．
- Ethernet ポート間の中継においては，100Mbps のスループットを実現．
- VPN における対向拠点は最大 1000 対地（拡張メモリ搭載時），ISDN 同時接続可能な対向拠点は 46 対地（PRI 拡張モジュール× 2 搭載時）と，大規模なネットワークに対応．
- 広域 Ethernet サービスや光アクセス，ADSL に対応する Ethernet 規格である 10/100Base-TX インターフェース 3 ポートを標準装備し，インターフェースを追加するための拡張スロットを 4 ポート装備．

■ 富士通（株）
価格：¥900,000 ～
TEL：03-6252-2660
E-mail：telecom@fujitsu.com

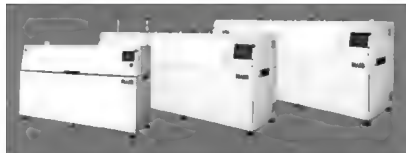
■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨 1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

HARD WARE

●鉛フリー対応リフロー炉

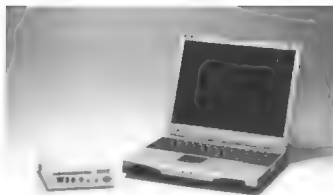
N_2 リフロー炉 エアリフロー炉

- 鉛フリーはんだリフローで，トップレベルのピーク温度ばらつきΔtと，最適な温度プロファイルを実現.
- S，M型でプレヒートゾーン数を5ゾーン化，L型で6ゾーン化など，総ゾーン数を8～10と多ゾーン化し，小型基板から大型基板まで安定した最適なプロファイルを実現.
- 独自のノズル「N_2リフロー炉：RNパイプノズル」，「エリアフロー炉：RAフラットノズル」を開発し，配置を「マトリックス熱風循環加熱方式」としたため，均一風速を実現し，熱伝送効率を促進.
- 「上下同一循環加熱方式」により，PCマザーボードのような大型基板でも，基板全体の均一加熱が可能.

■ 松下電工マシンアンドビジョン（株）
価格：オープン価格
TEL：06-6903-5129
E-mail：webmaster@naismv.co.jp
URL：http://www.naismv.com/

●ディジタルビデオストリーミングシステム

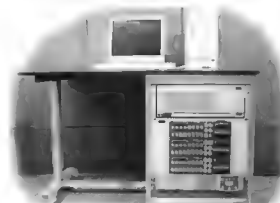
C9290

- 市販DVカメラなどのDV映像をDVNet ServerでDV-IPパケット変換を行い，LAN経由で配信し，Windows XPパソコン上のDVNetPlayerソフトウェアで再生.
- ネットワーク上での映像伝送時の問題となっていた，遅延やパケットロスに対して，高遅延ゆらぎ対応機能や断線復帰機能などで対応.
- DVNetServerは，使いやすいWebコントローラ機能により，IPアドレスなどの各種設定や配信コマンドの操作がリモートで行えるほか，再生ソフトウェアのインストーラもPCにダウンロード可能.

■ 浜松ホトニクス（株）
価格：オープン価格
TEL：053-584-0200　FAX：053-586-8467
E-mail：viewer@hq.hpk.co.jp
URL：http://www.hpk.co.jp/

●多チャネルデータ収集装置

model

- Windowsを搭載したPCで本格的なデータ収集システムを構築でき，計測がすぐに開始できるPC計測システム.
- アナログ入力は16チャネル単位で最大128チャネルまで拡張でき，リアルタイムでPCのHDDに無限連続収集が可能.
- A-D変換ユニットは16ビットで，全ユニットが同一クロックでサンプリング可能.
- 測定データは電圧値をはじめ，物理値で収録でき，バイナリファイルまたはテキストファイルで収集されるため，Excelなどでのデータ解析が可能.
- トリガ機能は内部の任意の3チャネル内のAND/OR条件が設置できるため，地震や台風などの自然環境の観測にも適する.

■ サンシステムサプライ（株）
価格：¥298,000～
TEL：03-3397-5241　FAX：03-3399-2245

●マルチプロトコルアナライザ

LE-7200

- 精度±0.01%の高速任意ボーレート対応技術を搭載.
- 最高2Mbps（全2重時）/4Mbps（半2重時）の通信を解析可能.
- 調歩同期（非同期）からBSC，HDLC，パケット通信まで多くのプロトコル解析機能を標準装備.
- RS-232-CとRS-422/485を標準装備し，X.21やRS-449などにも専用ケーブルのみで対応.
- マイクロドライブや大容量コンパクトフラッシュカードへ，最大1Gバイトの計測データを長時間連続記録.

■（株）ラインアイ
価格：¥480,000
TEL：075-693-0161　FAX：075-693-0163
E-mail：info@lineeye.co.jp
URL：http://www.lineeye.co.jp/

●ブロードバンドルータ

MR104X

- 最大92Mbpsの高速スループット（FTP）を実現.
- MIPS系CPU（BRECIS MSP2000），大容量フラッシュメモリ（2Mバイト）搭載.
- 家庭用ゲーム機，ホームサーバ接続用専用ポート（DMZポート）搭載で，家庭用ゲーム機，ホームサーバのネットワーク接続を実現.
- DMZポートに接続された機器はLANから物理的に切り離されるので，外部からの接続がLANに流れることなく，より高度なセキュリティを得られる.
- SPI，DoS攻撃防御，パケットフィルタリングなどの，高性能ファイアウォール機能を搭載.
- PPPoEマルチセッション対応により，異なるプロバイダに加入していても，専用サービスなどへの同時接続が可能.
- PPPoEアンナンバード対応により，プロバイダから複数のグローバルIPアドレスを取得して，LAN内に複数のサーバを設置可能.

■ オムロン（株）
価格：オープン価格
TEL：03-5435-2010
URL：http://www.omron.co.jp/ped-j/index.html

●音響/振動計測機器

NI PXI-4472B NI PCI-4474

- NI PXI-4472Bデバイスは，8チャネルダイナミック信号収録モジュールの新バージョンで，振動および低周波AC計測向けに最適化．4チャネルのNI PCI-4474ボードは，少ないチャネル数で24ビット分解能の精度を確保したい音響/振動のエンジニアリング分野に適する.
- デバイスはすべて24ビットのA-D変換器（ADC）を装備しており，110dBのダイナミックレンジを45kHzの帯域幅で提供.
- 4本または8本の同時サンプリングアナログ入力チャネルを装備しており，加速度計およびマイクロフォン用の内蔵型プログラマブル統合電子圧電(IEPE)調節機能を提供.
- LabVIEW音響/振動ツールセットやLabVIEW次数解析ツールセットなどと使用することで，1/Nオクターブ解析，オーダ解析/抽出などさまざまなテストを行うことができる.

■ 日本ナショナルインスツルメンツ（株）
価格：¥576,000～（NI PXI-4472B）
¥288,000～（NI PCI-4474）
TEL：03-5472-2970　FAX：03-5472-2977
E-mail：prjapan@ni.com

HARD WARE

●基地局テストセット

E7495A 基地局テストセット

- 送信機，受信機，アンテナケーブル，有線部分とのインターフェースなどのテスト機能のほか，オーバエアテストツールを装備し，基地局の設置，保守に必要な各種試験に1台で対応．
- 305 × 390 × 127mm，約9kgという小型，軽量化を実現．
- 電源を確保しづらい場所でも作業が可能な，バッテリ駆動を実現．
- 屋外でも見やすくするための大型画面を採用．
- シリアルポート，USB，Ethernet，PCMCIA，コンパクトフラッシュなど，各種インターフェースに対応．
- GSMや現在普及が進んでいるIS-95，cdma2000などの規格に対応．また，W-CDMAにも対応予定．

■ アジレント・テクノロジー（株）
価格：¥2,180,000～
TEL：0120-421-345

●Bluetoothシリアルユニット

PF-6070

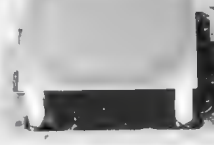
- Bluetoothソフトウェア開発キットと付属ハードウェアから構成される
- シリアルポートプロファイル（SPP）までを組み込み基板に移植済み．
- プロトコルスタックのライブラリとアプリケーションのソースコードが付属しているため，設定コマンドを追加したり，アプリケーションを追加することで，独自仕様の組み込みユニットを開発できる．
- 組み込み基板とデバッグ基板の回路情報も付属しているため，ハードウェアの試作は不要．
- デバッグ基板には，RS-232-Cレベルコンバータ，JTAG端子，I/Oポート，ステータスLEDを搭載．

■ キヤノンアイテック（株）
価格：¥500,000
TEL：042-366-1161　FAX：042-366-8844
E-mail：sales@citech.co.jp
URL：http://www.citech.co.jp/

●非接触カードIDリーダ

RCR1

- 非接触カードType Cの，固有IDを読み取るカードIDリーダ．
- 固有ID読み取りのため，同一仕様のカードであれば，カード用途に依存することなく使用可能．
- 読み取り距離は，約10mm．
- 読み取りデータ出力はRS-232-Cで，速度は9600bps．
- 電源はDC5V ± 5%で，約100mA．
- 外形サイズは，約75 × 55 × 28mm．

■（有）ビーアイティ
価格：¥26,800
FAX：03-3635-9094
URL：http://homepage3.nifty.com/co_bit/

●IPコア

CorePCIX

- 高速アクセラレータFPGAに最適化された，133MHzで動作するIPコア．
- 組み込みアプリケーション向けのPCI-X専用のソリューションとして，またはPCI-X仕様1.0a版に準拠したアドインカードアプリケーション向けソリューションとして利用可能．
- 32ビットと64ビットのマスタおよびターゲット機能，64ビット転送，PCI-X/PCIバースト処理命令機能を提供．
- シンプルなローカルバスインターフェースへの追加ロジックの統合や，Webベースの構成インターフェースを使ってこのコアのパーソナライズと未使用ロジックの削除が可能．

■ アクテルジャパン（株）
価格：$18,000（シングルユースネットリスト）
$1,500（評価基板ベース）
TEL：03-3445-7671　FAX：03-3445-7668
URL：http://www.actel.com/

●組み込みソリューション

GR-XCBASE

- 「GR-XFUNC」，「GR-XCTL」，「GR-XCOM」の3種類の基盤機能を提供する，携帯，家電機器向けの組み込みソリューション製品．
- シンプルなローカルバスインターフェースへの追加ロジックの統合や，Webベースの構成インターフェースを使ってこのコアのパーソナライズと未使用ロジックの削除が可能．
- シンプルなローカルバスインターフェースへの追加ロジックの統合や，Webベースのインターフェースを使ってこのコアのパーソナライズと未使用ロジックの削除が可能．「GR-XFUNC」は，携帯，家電機器の機能拡張/連携基盤で，拡張機能登録，拡張メニュー表示，USB機能拡張モジュールによる機能拡張，インターネット連携による機能拡張などを提供．
- シンプルなローカルバスインターフェースへの追加ロジックの統合や，Webベースの構成インターフェースを使ってこのコアのパーソナライズと未使用ロジックの削除が可能．

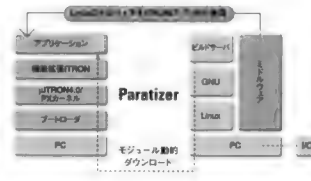
■（株）グレープシステム
価格：下記へ問い合わせ
TEL：045-222-3754　FAX：045-222-3759
E-mail：info@solutions.grape.co.jp
URL：http://www.grape.co.jp/

●μITRON開発環境

Paratizer

- パソコンとオープンソースソフトウェアだけで，μITRON用のアプリケーションやデバイスドライバの開発が可能となるシミュレーション開発環境．
- パソコンのCPU上で直接稼動するμITRON4.0カーネルをベースとしている．
- 任意のタスクやデバイスドライバだけを，動的にダウンロードして実行することが可能．
- 実行環境をリセットすることなく，デバッグ対象オブジェクトだけをダウンロードするため，デバッグ時間を短縮することが可能．
- メモリ保護機能により，不正アクセスを検出することができる．

■（株）エーアイコーポレーション
予価：¥100,000以下（50台時）
TEL：03-3493-7981　FAX：03-3493-7993
E-mail：sales@aicp.co.jp

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　　（編集部）

SOFT WARE

●Javaパフォーマンス保証ツール

JProbe Suite 5.0J

- Javaシステムに潜む問題を検出するツールで，高レベルな計測機能と分析ウィンドウを併せ持つ．
- 効率の悪いロジックやメモリを著しく消費するオブジェクト，非スレッドセーフな処理などを簡単な操作で特定可能．
- 2回のマウスクリックでメモリ問題の分析に必要なデータの収集が可能．
- オブジェクトの生成数に連動したアラートやメモリリークの発見支援機能が加わり，多角的な分析が可能．
- アプリケーションのパフォーマンス計測と，ソースコードのカバレッジ解析のスピードが，前バージョンと比較して3～5倍に向上したことで，規模の大きいJ2EEシステムでもストレスなく利用可能．
- リモート分析の結果確認に必要だった設定が不要になり，サーバとリモートコンピュータ間で共有フォルダを介する手間を省き，双方で煩雑な設定を行うことなく，分析結果を直接リモートコンピュータにダウンロード可能．

■ グレープシティ（株）
価格：¥368,000
TEL：048-222-3001　FAX：048-222-1211

●Java開発ツール

Formula One e.Spreadsheet Engine Formula One e.Report Engine

- 米国Actuate社が開発したJavaツール．
- Java上でMicrosoft ExcelファイルとPDFファイルの生成，編集，保存が可能．
- データベース，XML，テキストファイルにアクセスし，自動的にデータの表示形式を整え，計算，分析する．
- スタティックなデータをCSVやExcelに書き込む以上のことが可能で，動的に多彩な機能をもったExcelファイルの作成ができる．
- GUIを使用しないサーブレット，JSP，EJBへの埋め込み可能なサーバサイドスプレッドシートエンジンとしても，GUIを使用するJavaデスクトップアプリケーションでのExcel互換のグリッドとしても使用可能．
- All-Java認定APIにより，簡単で自由度の高いメソッドを使用してデータベース，テキストファイル，XMLなどのデータにアクセスし，ExcelやHTMLレポートの配信が可能．
- Formula One e.Report Engineは，PDF，XML，DHTMLまたはHTML形式でレポートを配信．

■ エクセルソフト（株）
価格：¥750,000
TEL：03-5440-7875　FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com
URL：http://www.xlsoft.com/

●CTIアプリケーション開発ツール

VBVoice5.0

- Visual Studio.NETに対応し，Visual BASIC以外にVisual BASIC.NET，Visual C#などの.NETの開発環境を選択することが可能となり，開発したCT（コンピュータテレフォニー）プログラムは，.NET Framework上での実行が可能．
- 大規模CTソリューションを構築するための新しい分散処理アーキテクチャを導入している．そのため，個々の独立したプログラムをネットワーク上のそれぞれのノード上で実行し，分散処理管理機能によって，一つの統合したアプリケーションとし，通話中の異なるノードへの引き継ぎ処理を心配する必要がない．
- VoIPのプロトコルとしてH.323をサポートしており，Dialogic IP-Link音声処理ボードを利用して，従来のCT開発手法でVoIPをサポートしたCTプログラムの開発が可能．
- 標準パッケージでは，Add-on製品以外のすべての機能が利用可能．

■（株）プロトン
価格：¥248,000
TEL：03-5337-6431　FAX：03-5337-6130

●開発ツール

インテル数値演算ライブラリ 6.0

- インテルプラットホーム上で高いパフォーマンスが求められるアプリケーションに最適化された，数学関数一式を提供．
- 7次元までの混合基数離散フーリエ変換をサポート．
- 1回の呼び出しで，複数の1次元変換処理を実現．
- ベクトル統計ライブラリ（VSL）による高度な乱数作成機能を搭載．
- 強化されたプロセッサスタティックライブラリは，ランタイムにCPUを検知し，最適化されたコードを実行．
- インテルItanium2プロセッサを含む，最新のプロセッサを搭載しているシステム上でのパフォーマンスを向上．
- Windows版とLinux版を提供．
- インテルFortranコンパイラ6.0以上，Compaq Visual Fortran 6.0，インテルC++コンパイル6.0以上，Microsoft Visual C++6.0以上，GNUコンパイラをそれぞれサポート．

■ エクセルソフト（株）
価格：¥28,000
TEL：03-5440-7875　FAX：03-5440-7876
E-mail：intel@xlsoft.com
URL：http://www.xlsoft.com/intel/

●アプリケーションサーバ

Borland Enterprise Server 5.2 日本語版

- 業界標準のCORBA技術「Borland Visi Broker」を基盤とし，高いオープン性と拡張性を実現したハイパフォーマンスアプリケーションサーバ．
- 次世代インターネットプロトコル仕様であるIPv6に対応．
- Apache Webサーバ2.0とTomcat Webコンテナ4.1に，ボーランドの分散テクノロジを統合し，高い拡張性と可用性を備えたWebアプリケーションの実行をサポート．
- Web層のボトルネックとなるサーブレット，JSPの運用環境を多重化し，負荷分散できる．
- Apache AXISテクノロジをベースとしたWebサービスサーバを搭載することで，最新のWebサービスアプリケーションの開発，運用，公開，利用が可能．
- 既存のJavaクラス，EJB，CORBAアプリケーションをWebサービスとして公開できるため，最新テクノロジによるビジネス統合をすばやく実行可能．

■ ボーランド（株）
価格：¥68,000～
TEL：03-5350-9358　FAX：03-5350-9387
URL：http://www.borland.co.jp/

●プロトコルスタック

Nucleus USB

- 多種多様なUSBデバイスをホストする，USBデバイスの作成，USBハードウェアコントローラを動作させるための組み込みソフトウェアで構成．
- 一般的なアプリケーション要求に応えるアウトオブボックスソリューションを提供する，USB互換クラスドライバをオプションとして備えており，標準USBデバイスを短時間で作成することが可能．
- ホストとデバイスコンポーネントの両方を備えており，Nucleus USB Hostはセットトップボックス，POS端末，計測器，ゲームコンソールおよびPDAなどのUSBデバイスがプラグインされる組み込みシステムの開発時に効果を発揮する．
- Nucleus USB Deviceは，ジョイスティック，カメラ，スキャナやプリンタ，ハードディスクドライブやその他ストレージメディア，ルータやその他通信デバイス，PDA，電話や音楽プレーヤなどのホストにプラグインするUSBデバイスの構築向け．

■ メンター・グラフィックス・ジャパン（株）
価格：下記へ問い合わせ
TEL：03-5488-3041　FAX：03-5488-3032

SOFT WARE

●Linux OS

LindowsOS

- 米国Lindows.com社が開発した，Linuxをベースとしたオペレーティングシステム.
- 一定料金で，あらゆる種類のアプリケーションソフトウェアが1年間，無制限にインストールおよび利用が可能.
- 数回のマウスクリックだけで，ほとんどのアプリケーションソフトウェアのダウンロードとインストールが可能.
- 独自のClick-N-Runテクノロジにより，アプリケーションソフトウェアのダウンロード配信を完全ディジタル化し，開発者のソフトウェアを公開して，流通の手間とコストを大幅に削減.
- オフィスアプリケーションから，インターネット，マルチメディア，ゲーム，開発ツールにいたるまで，多数のアプリケーションソフトの利用が可能.
- マウス，キーボード，ウィンドウなどを用いた直感的でわかりやすいユーザーインターフェースを装備.

■ エッジ（株）
価格：下記へ問い合わせ
TEL：03-5766-7211
E-mail：lindows@edge.jp

●コンプライアンステストソフトウェア

TDSDVI

- 同社のオシロスコープ（TDS7000シリーズ）にインストールされ，高速かつ高い信頼性による規格適合試験が可能となる.
- メニューボタンを押すだけで，DVI規格で必要とされる試験が実施され，報告書を作成.
- トランスミッタ，ケーブルおよびレシーバなど，さまざまな部品の試験において，アイダイヤグラム，ジッタ，スキュー，立ち上がり時間，立ち下がり時間の測定が可能.
- ユニットインターバルを計算し，100万回におよぶ波形取り込みを行い，10個のピクセルにわたってワーストケースのアイダイヤグラムテストを行うことで，信号品質が詳しくわかる.
- オシロスコープは，テスト項目に応じて自動的に設定され，マスクも生成.
- 接続試験では，詳細な測定結果からパス/フェイルを表示.
- 2種類の異なった機器の接続試験の結果比較も可能.

■ 日本テクトロニクス（株）
価格：¥288,000
TEL：03-3448-3010　FAX：0120-046-011
URL：http://www.tektronix.co.jp/

●システムユーティリティ

Acronis システムユーティリティシリーズ

- 米国Acronis社が開発した，システムユーティリティ製品.
- Acronis TrueImage 6.0は，ハードディスクのバックアップイメージをウィザードに従って操作するだけで，パソコンの環境を丸ごとバックアップ/リストア可能なツール．独自のテクノロジにより，イメージ作成をバックグラウンドで行う.
- Acronis PartitionExpert 2003は，データを保持したままパーティションを操作できるツール．削除してしまったパーティションをリカバリする「RecoveryExpert」が付属.

■（株）プロトン
価格：¥5,800～¥14,800
TEL：03-5337-6432　FAX：03-5337-6130
E-mail：ps@sb.proton.co.jp
URL：http://softboat.jp/

●ウィルス検知ソフト

PestPatrol

- ペストパトロール社が開発した，ハッカーツール，スパイウェア，トロイの木馬などの不正アクセスツール（ペスト）を検出し，隔離/削除するツール.
- 12,000以上のペストファミリから，68,000以上のペストを検出可能.
- ファイル，メモリ，レジストリ，起動部分についてマニュアルおよびリアルタイムでスキャン可能.
- 検出したペストに対して，削除または隔離処理が可能.
- ネットワーク管理，セキュリティ検査，監査など，管理者に必要なプログラムの除外が可能.
- 個人情報を漏洩する危険のあるスパイウェアクッキーの自動削除.
- 既知および未知のキーロガーの検出が可能.
- ログインスクリプトにより，容易にクライアントにインストールでき，自動更新が可能.

■ 富士マグネディスク（株）
価格：下記へ問い合わせ
TEL：042-314-6602　FAX：042-314-6610
E-mail：fmdinfo@fmd.fujifilm.co.jp

●ウィルス検出＆駆除ソフト

NOD32 アンチウィルス

- スロバキアのイースト社が開発した，ウィルス対策ソフト.
- ファイルオープン時，実行時，新規作成時，名前変更時などで常にパソコンを監視し，あらゆるファイルのウィルス検査を行う.
- もっとも侵入の可能性が高いメール受信プロトコル「POP3」に対しても監視を行う.
- ウィルスの検出は，シグネチャ方式と独自開発のヒューリスティック方式で実施することで，高い検出率と高速スキャンの双方を実現.
- ヒューリスティックエンジンは，2種類のアプローチで高いウィルス検出率を実現.

■ キヤノンシステムソリューションズ（株）
価格：¥6,800（パッケージ版）
¥4,000（ダウンロード版）
TEL：03-5815-7258
E-mail：nod-info@canon-sol.co.jp
URL：http://canon-sol.jp/

●ネットワークアナライザソフトウェア

Observer Version8 Expert Observer Version8

- 米国ネットワーク インスツルメンツ社が開発した，ネットワーク上を流れるデータを収集し，分析を行うためのツール.
- 有線LANの10Base-T/100Base-TXおよび，IEEE802.11a/b無線LANに対応．IEEE 802.11gへの対応も予定している.
- オプションのProbeを利用すれば，ローカルなセグメントだけでなく，遠隔地のセグメントも監視できる.
- 500種類以上のプロトコルを解析可能.
- さまざまな角度からネットワークの状態をリアルタイムで統計，グラフを表示.
- 熟成されたユーザーインターフェースで，直感的な操作が可能.
- Expert Observerは，Observerの全機能に加えて障害解析に役立つエキスパート機能を搭載.

■ コマツ
価格：¥150,000（Observer）
¥400,000（Expert Observer）
TEL：045-411-2701
URL：http://www.komatsu.co.jp/el/lan/

■弊誌では新製品に関するニュースリリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

海外・国内イベント/セミナー情報
INFORMATION

海外イベント

7/14-16 **SEMICON West 2003**
Moscone Center, San Francisco, CA, USA
SEMI
http://events.semi.org/semiconwest/

7/17-19 **NEPCON Thailand 2003**
Bangkok International Trade & Exhibition Centre, Bangkok, Thailand
Reed Exhibitions
http://www.nepconthailand.com/index.php

7/27-31 **SIGGRAPH 2003**
San Diego Convention Center, San Diego, CA, USA
SIGGRAPH
http://www.siggraph.org/s2003/

7/30-31 **EmbeddedSystems Conference Asia**
Taipei International Convention Center, Taipei, Taiwan
CMP Media Inc.
http://esconline.com/asia/

8/4-7 **Linux World Conference&Expo**
Moscone Convention Center, San Francisco, CA, USA
IDG WORLD EXPO
http://www.linuxworldexpo.com/linuxworldny03/V40/index.cvn

8/17-19 **HOT CHIPS 15**
Stanford Memorial Auditorium, Palo Alto, CA, USA
IEEE
http://www.hotchips.org/

8/20-22 **Hot Interconnects**
Stanford University, Palo Alto, CA, USA
IEEE
http://www.hoti.org/

国内イベント

6/25-27 **設計・製造ソリューション展**
東京国際展示場(東京ビッグサイト，東京都江東区)
リードエグジビションジャパン
http://web.reedexpo.co.jp/dms/

6/30-7/4 **NETWORLD+INTEROP 2003 TOKYO**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
Key3Media
http://www.interop.jp/index.php

7/9-11 **組込みシステム開発技術展 ESEC**
東京国際展示場(東京ビッグサイト，東京都江東区)
リードエグジビションジャパン
http://web.reedexpo.co.jp/ESEC/jp/

7/9-11 **YRP移動体通信産学官交流シンポジウム**
横須賀リサーチパーク(神奈川県横須賀市)
YRP研究開発推進協会 独立行政法人通信総合研究所
http://www.yrp.co.jp/event/aig/2003/

7/15-18 **InterOpto'03**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
OITDA
http://www.oitda.or.jp/

7/16-18 **WIRELESS JAPAN 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
リックテレコム
http://www.ric.co.jp/expo/wj2003/index.html

8/5-8 **Microsoft Tech ・ Ed&EDC 2003 YOKOHAMA**
パシフィコ横浜(神奈川県横浜市)
マイクロソフト
http://www.event-info.jp/te03/default.htm

開催日，イベント名，開催地，問い合わせ先の順

セミナー情報

Eclipse活用実践講座
開催日時　：6月30日(月)～7月1日(火)
開催場所　：(株)コメット　初台トレーニングセンター(東京都渋谷区)
受講料　：62,000円
問い合わせ先：(株)カサレアル技術教育部，☎(03)5791-5066，FAX(03)5791-5067
http://www.casareal.co.jp/j_school/eclipse.html

オブジェクト指向プログラミング・フリーソフト
Squeakプログラミング入門[準備編]
開催日時　：7月1日(火)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　：48,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_165.htm

無償ツールによるSystemCデザインのハード/ソフトの検証と合成
開催日時　：7月2日(水)～4日(金)
開催場所　：BIZ新宿(東京都新宿区)
受講料　：198,000円
問い合わせ先：(株)礎デザインオートメーション営業部，☎(03)6762-1471
http://www.ishizue-da.co.jp/

画像処理/計測のための画像解析技術
開催日時　：7月8日(火)～9日(木)
開催場所　：高度ポリテクセンター(千葉県千葉市)
受講料　：20,000円
問い合わせ先：雇用・能力開発機構 高度ポリテクセンター事業課，☎(043)296-2582　http://www.apc.ehdo.go.jp/

次世代コンピュータ・アーキテクチュア自律コンピューティング入門技術解説
開催日時　：7月9日(水)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　：48,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_119.htm

SIP(プロトコル)入門/概要・インプリメント・セキュリティ・品質
開催日時　：7月14日(月)
開催場所　：アドバンスト・テクノロジーセンター(東京都千代田区)
受講料　：44,100円
問い合わせ先：(株)アドバンスト・テクノロジーセンター，☎(03)3518-6441，FAX(03)3518-6147　http://www.at-center.co.jp/pdf/A_1284.pdf

入門Linuxデバイスドライバ開発技法
開催日時　：7月14日(月)～15日(火)
開催場所　：オームビル(東京都千代田区)
受講料　：62,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c030714a2.htm

TCP/IP組込み設計手法入門
開催日時　：7月16日(水)
開催場所　：アドバンスト・テクノロジーセンター(東京都千代田区)
受講料　：43,050円
問い合わせ先：(株)アドバンスト・テクノロジーセンター，☎(03)3518-6441，FAX(03)3518-6147　http://www.at-center.co.jp/pdf/B_2116.pdf

ARM9/SH-4デザインキットを使った組み込み開発
開催日時　：7月17日(木)
開催場所　：ガイオ・テクノロジー(株)日本橋営業所(東京都中央区)
受講料　：無料
問い合わせ先：ガイオ・テクノロジー(株)，seminar@gaio.co.jp，☎(03)3662-3041

IEEE802.11a/g無線LANの標準化動向とシステム構築技術
開催日時　：7月18日(金)
開催場所　：オームビル(東京都千代田区)
受講料　：52,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c030718n.htm

TCP/IPによるI/O制御の実際～Ethernetを利用した組み込み機器の設計
開催日時　：7月18日(金)～19日(土)
開催場所　：CQ出版セミナールーム
受講料　：25,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

Windowsプログラマのための PocketPCプログラミング入門
開催日時　：7月22日(火)～7月23日(水)
開催場所　：DISパソコンスクール(東京都文京区)
受講料　：98,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109　http://icp.hicorp.co.jp/seminar/c-vc/pocketpc.asp

電子回路の高速化に対応する基礎技術
開催日時　：7月24日(木)～25日(金)
開催場所　：ホテル機山館(東京都文京区)
受講料　：61,750円
問い合わせ先：サイペック(株)，info@r-sipec.jp　http://www.rlz.co.jp/

計算力学の基礎コース
開催日時　：8月18日(月)，19日(火)，20日(水)，25日(月)，26日(火)，27日(水)　計6日間
申し込み締め切り日：7月29日(火)
開催場所　：かながわサイエンスパーク(神奈川県川崎市)
受講料　：72,000円
問い合わせ先：(財)神奈川技術アカデミー，☎(044)819-2033，FAX(044)819-2097　http://home.ksp.or.jp/kast/

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

IPパケットの隙間から

脅迫と訴訟

58

祐安重夫

知人(女性)のところに突然，知らない人物から電話がかかってきて，関西弁で借金を返さなければこれから殺しにいくと脅されたそうだ．借金についてはまったく身に覚えがなかったので，警察に通報したところ，すぐに警官が数人でやってきて，同じ所轄の管内ですでに3件，同じ携帯電話から同様の脅迫電話がかかってきていると教えてくれたらしい．脅迫電話を番号通知でかけるのも相当間抜けな話だが，他人の携帯電話かプリペイド携帯を使用していたのだろう．

その後，知人のところには怪しい人間は訪ねてこなかったようだが，単なる嫌がらせなのか，あわよくば本気で金を取ろうと思っていたのか，よくわからない．

他人をだまして金を取ろうという手口には，単純だが，ついだまされそうになるものがある．よくあるのは，電話帳に広告を掲載している会社宛に，NTTでもないのに広告料の請求書を送るというのがある．これにだまされて，つい広告料を払ってしまううっかりした人がいて，こういう詐欺は採算がとれているようだ．

そういえば筆者のところにも，「行政なんとか」といった名前(正確な名前はおぼえていない)の小冊子が毎月送られてきていたが，内容に興味がないので封も切らずにそのままゴミ箱に直行させていた．すると1年ほどして，購読料の請求書がやってきた．もちろんこの請求書もゴミ箱に直行させたところ，しばらくしたら小冊子も来なくなった．これもうっかり払ってしまった人が，結構いそうである．

総会屋のような悪辣なものは，法による規制ができて，もしかすると地下に潜って行われているのかもしれないが，表面的には減少しつつあるようだ．しかし，訴訟王国のアメリカあたりでは，特許権や著作権を盾に他の企業を提訴したり，実際に提訴はしないまでも，提訴すると通告して合法的に和解金を稼ごうとする会社もあるようだ．

さて，ここまでの話とはまったく関係がないのだが，3月にSCO GroupがIBMを提訴したときは，ちょっと驚いた．そのときには事態がそれほど重要だという認識がなかったが，IBMがLinuxに参入したことでLinuxの普及にはずみがついたのは確かだとしても，IBMの技術導入がなければLinux自体が現在の技術水準に到達できなかったかのような主張は，Linuxやオープンソースの開発者を馬鹿にしたものであり，それは逆にSCO Groupの馬鹿さ加減の現れであった．

そもそもSCO GroupにIBMを超える，あるいはLinuxやオープンソースの開発者を超える技術力やノウハウがあったとは，とうてい思えない．現在にいたるもSCO Groupは，コード盗用の証拠をまったく示していない．

しかし5月になって，この問題がLinuxコミュニティ，あるいはオープンソースコミュニティ自体に対する提訴の可能性へと変化してきたとき，これは他人事ではなくなってきた．Linuxを仕事で使用し，いくつかのサーバを管理している筆者にとっては，とんでもなく迷惑な営業妨害である．SCO Groupが全世界の1500社に，「知的所有権の侵害の可能性」についての書簡を送ったことなど，ほとんど嫌がらせに近い印象を受ける．ちなみにこの書簡は，日本の企業にも送られているが，SCO Groupの日本法人には，まったくそのことは知らされていなかったという．

ここで面白いのはNovellが登場して，UNIXについてのおもな著作権や特許を所有しているのは自分達であり，SCO Groupではないと反撃し始めたことだ．この発表によって，SCO Groupの株価は24%下がったそうだが，それ以前にIBMを提訴して以来，SCO Groupの株価が10倍近くまで上がっていたことのほうが驚きである．

このNovellの主張が正しいとすれば(たぶん間違いないと思うが)，SCO GroupはIBMを契約条項違反で訴えることは可能だとしても，他のLinuxディストリビュータやユーザーを訴えることは困難だということだろう．Novellは特許や著作権を盾に他者を訴えることはしないと言明しているようだし，もし訴えを起こしたらどうなるかは，現在のSCO Groupがオープンソースコミュニティだけではなく，コンピュータコミュニティ全体からどう評価されているかを見れば明白である．

すでにコンピュータとネットワークの標準的なインフラストラクチャとして大きな位置を占めているLinuxに対して，いまや過去の遺物となりつつあるUNIX System Vの権利をもつ(と自分たちは考えている)企業による悪あがきというのが，この一連の騒動に対する印象である．

ところでこの騒動の中で，SCO Groupとわざわざライセンス契約を結んだ企業がある．Linuxやオープンソースのコミュニティを異常に敵視してきたこの企業(どことはいわないが)の今回の行動は，Linux攻撃のための戦略の一貫であり，ライセンスが必要だったからではないという噂が，さまざまな場で飛び交っているのは，火のないところに煙は立たずという諺を，地でいっているかのようだ．

話は最初に戻るが，電話で意味不明の脅迫を受けた知人は，その後なんの被害も受けていないようだ．悪が栄えたためしはない．

すけやす・しげお　インターメディアアクセス

読者の広場

2003年6月号特集「TCP/IPの現在とVoIP技術の全貌」に関して

▷VoIP技術についてシーケンスも含めて詳しく書かれていてたいへん参考になりました．今後はインターネットの速度関係と故障時の切り分けなどの記事も期待します．（マーチャリ）

▷IP電話に関する記事はたいへん興味深く，参考になりました．また，自分でIP電話が作れるなんて楽しそうですね．チャレンジしたくなる内容です．たいへん理解しやすく，興味深く読みました．（福沢陽介）

［編］VoIPの各種プロトコルは公開されているうえに，それを実装したソフトウェアも公開されています．自分でそれらを実際に試してみると理解も深まり，良いかもしれませんね．

▷特集第5章の『VoIPにおけるセキュリティ』が興味深かった．インターネットと同じ可能性があるのは当たり前なのですが，一般の利用者はこれまでの電話と同じと考えるでしょうから，より効果的な対応をほどこす必要があるかもしれないと感じました．（玉出のタマ）

［編］今までの電話ではセキュリティのことなどはまったく考慮する必要はなかったのですが，VoIPではそうは行きません．セキュリティ確保のためには専用ソフトが必要となり，追加投資が迫られることがありますが，それを「当たり前のもの」と受け止める「意識改革」が必要なのかもしれません．

▷VoIPはユーザーの立場で使用することもまれにあり，興味がありました．技術的な知識がほとんどなかったので，今回の特集は非常に勉強になりました．（JR9JUK）

▷今月のVoIPは良かった．全般的にわかりやすかった．ただQoSのサービスを提供したいのだが，この部分の説明がもっとあってもよかったと思う．（岸田昌也）

▷VoIPがVoice over IPの時代は短いかと．将来はVideo over IPになると思います．現在翻訳電話は特定の業務内容に対応可能といった状態ですが，携帯電話のように特定話者の前提であれば認識率も高く，途中はまさにコードで送れば，通信データ量は減ります．そしてテキスト読み上げソフトに話者の特徴を加えれば……．（川名一）

▷音声でメール入力して，そのメールを音声で聞くという環境も遠くなさそうですね．（麻由美）

▷特集を楽しく読ませてもらいました．個人的な興味はとてもあり，楽しめました．ただ，現実は……会社でもIP Phoneとか騒いでいる人もいたりする状態です．それはそれでいいのですが，何度も説明したのに，ブリッジなしの10Mbps環境なので，導入当初からトラブル続きで，今後も期待薄でしょう．p.105の内容を社内で宣伝してまわりたくもなりますが，まぁとりあえず距離を置いています．（ハ）

［編］これを機にギガビットEthernetなどを導入して，景気回復に貢献してみてはいかがでしょうか（回線の速さとレスポンスは厳密には違うのですが）．

その他

▷「家電機器をネットワーク化するアーキテクチャUniversal Plug ans Playの全貌」に興味をもちました．私は現在，液晶で有名な某メーカーで通信関連のシステムに携わっているので，そのうちきっと役に立つときが来るでしょう．連載は残す価値があると思います．

それにしても，組み込み系システムの現場は女性がいない．（棄婚者）

［編］弊社ではPDF版Interfaceも発行しております．気に入った連載をプリントしまとめて私家本にすることもできますので，手元に置かれてみてはいかがでしょう．

特集担当デスクから

☆「最近のIF誌の特集記事の内容を，より深く読み込み/活用するための用語解説」がコンセプトの特集をお送りします．大まかなコンセプトは，本誌2001年4月号特集「現代エレクトロニクスの基礎知識」を継承しています．このときの読者の方からのさまざまなフィードバック，編集側の反省などを加味し，さらに役立つ用語解説特集をめざしたのですが，いかがでしょうか？

☆二つの特集の間に約2年弱の時間が流れています．扱うテーマは微妙に変わっています．たとえば今回のICカード，ARM，BSD，データベース，シミュレーションなどは，前回，表立って出てきてはいません．

☆ある筆者から「Webでデータ入力のしくみを作って用語集を作り込めば，あとは更新していくだけで効率的じゃないの？」と提案いただきました．「究極の用語集」は，パソコンなどから簡単に呼び出せ，キーワード検索・AND/ORなどのオプション検索機能をもち，ハイパーテキスト化されていて関連用語にすぐジャンプでき，より詳細なコンテンツへのリンクもたどれる，そしていつでも更新できるような，「生きた用語集」かもしれません．読者の方により役立つ技術情報にすべく，いろいろ挑戦していきます．

☆Linuxについては，予告にあげておきながら誌面の都合でとりあげきれませんでした．申し訳ありません．次月号で，フレッシャーズ向け特設記事として掲載予定です．

読者の広場

▷ XPortの記事が参考になった．実験のために簡単な機器で振動スペクトロメトリを取れないか検討しており，ちょうどADXL202のことも調べていたところで，すごくタイムリに思えた．当然，ネットワーク接続（とくにWebサーバ機能を通した通信）に興味がある．後編に期待している．（萩）

▷ TCP/IPからSIPまで幅広く解説されていて非常に役立った．保存版にします．またUPnPも知りたかった技術の一つなので，来月号も買います．Interfaceらしい軽めの読み物も増やしてほしいです．（KAZZU）

アンケートの結果

興味のあった記事（2003年6月号で実施）

①第2章 VoIP技術の基礎知識
②第1章 TCP/IPの基礎と現状
③第6章 Gphone ――ソフトウェアが電話になる時代
④第3章 SIPを用いたシグナリングの実際
⑤第4章 VoIPで用いられる音声CODECの詳細
⑥第5章 VoIPにおけるセキュリティ
⑦第7章 オープンソースで作るIP電話
⑧ UPnPの全貌（第1回）
⑨フリーソフトウェア徹底活用講座（第10回）
⑩Webサーバ機能をもつEthernet-シリアルコンバータ「XPort」活用技法（前編）
⑪シニアエンジニアの技術草子（弐拾八之段）
⑫ハッカーの常識的見聞録（第30回）
⑬ Show & News Digest
⑭開発環境探訪（第19回）
⑮ IPパケットの隙間から（第56回）
⑯ Engineering Life in Silicon Valley（対談編）
⑰Microwindowsを使った組み込み向けGUIプログラムの作成事例（応用編）
⑱ XScaleプロセッサ徹底活用研究（第1回）
⑲ IP.net JAPAN 2003

特集『TCP/IPの現在とVoIP技術の全貌』についてのアンケートの結果

Q1 VoIP技術に関して，興味のある分野はどこですか？（複数回答可）

①基礎技術全般（29%）
②シグナリング（SIP）（11%）
③シグナリング（H.323）（3%）
④ネットワーク構築（6%）
⑤セキュリティ（18%）
⑥ハードウェア（9%）
⑦実装例（3%）
⑧ビジネスモデル（11%）
⑨その他（9%）

Q2 すでにVoIPアプリケーションを使っていますか？

①勤務先と自宅で使っている（8%）
②勤務先で使っている（0%）
③自宅で使っている（8%）
④使っていない（83%）

Q3 現在注目しているネットワーク技術は何ですか？

P2P，IPsec，UPnP，VoIP，IPv6，無線LAN，Bluetooth，帯域保証，QoS

読者プレゼント

●**応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2003年7月30日（必着）までにご応募ください．なお当選者の発表は発送をもってかえさせていただきます．**

（1）マグカップ　**（5名）**

アンカーシステムズ（株）
（http://www2.noritz.co.jp/anchor/）

Interface

次号予告

2003年9月号は
7月25日発売です

C/C++言語による ハードウェア設計入門

ディジタル回路とSystemCの基礎/組み合わせ回路のSystemC記述/順序回路とステートマシンのSystemC記述/SystemCを用いたハードウェア・ソフトウェア協調設計/SystemCを用いたシステム設計の実際

これまでハードウェア設計に使用されてきたHDL言語に代わり，C/C++言語をハードウェア設計に適用することに注目が集まってきた．これらハードウェア設計用C/C++言語は，ANSI-C/C++をベースとし，それらにハードウェア設計向け記述を拡張したものとなっているため，すでにC/C++言語を修得済みのプログラマにとっては理解しやすい．また，ハードウェアとソフトウェアを別々に開発せず，システム全体として設計する「ハードウェア/ソフトウェア協調設計」が可能になるという利点もある．

そこで今回の特集は，システム記述言語SystemCを用いて，実際にC/C++言語によるハードウェア設計を修得する．

★次号には，記事関連ファイルなどが満載されたCD-ROM『InterGiga No.31』が付属します！

編集後記

■2002年8月号の本欄では「仕事用PC突然電源OFF事件」(?!)の渦中を記した．今のところ，壊れたPCであたふたしていないだけLucky？閑話休題，「風薫る5月」がすぎようとしています．汗っかきの私は，朝の電車でもスーツを網棚に．が，冷房を効かせすぎの電車もあり，冷房との調整，各所で気をつけねば．（洋）

■情報を受け取ると，それが使えるのか，使えないのかの判断をまず行わなければならない．そして使えると判断した場合，より深く理解するための行動が必要となる．情報が入ってくる限り，この無限ループは続く．ちょっとの間，それを全部止めてみたところ，つかの間の安らぎを得ることができた．（＝IO）

■DVDの再生中，音声を切り替えようとメニューボタンを押したんですが，画面が切り替わらず，プレーヤからはピックアップシークのリトライ音が……．最近はDVDのオーサリングも安定してきたと思ってたんですけどネ(昔，再生途中で画面が崩れて止まるソフトがあった^^;)．新しいHDD＋DVDレコーダを買えと？（M）

■Aを勉強するためにはBという知識が前提とされるので，先にBを勉強しようとするとそこではCという知識が前提とされ，Cを(以下略)．日々是決戦の日常の中で，勉強する時間を捻出することは大変だと思いますが，上を目指すためには一段ずつ階段を上っていかなければならないわけで……一生勉強なんだなぁ．（み）

■半導体メーカーに元気がないこともあり，元気のよさそうな自動車メーカーに注目が集まっている．確かに，トヨタやホンダは最高益を記録し，日産も驚くべき復活を成し遂げたが，今後も安泰かどうかはわからない．米国も日本のバブル時代に，ビッグ3は今とは比較できないほど好調だった．米国の現在は将来の日本と見て，まず間違いないと思うが．（Y）

■猫好きか犬好きかと聞かれれば私は猫好きです．子供の頃から飼っていたからかも．猫は「わがまま」で呼んでも来ないとも言いますが，実家にいる猫は遠くにいても呼べば犬のように帰ってきます(笑)．とてもかわいいですよ．今は猫がいない生活なのでちょっと寂しいです．（Y2）

■イラク復興が心配である．一部の報道によれば反米意識は高まり，無法地帯化しつつあるという．そもそもフセイン政権が崩壊した中で，イラク国民というものは存在しているのだろうか．そんな中に自衛隊を派遣するであろう日本政府はどの程度の危機意識を認識しているのか．（ちゃん）

■世の中には永遠に生き続ける生物がいるらしい．ベニクラゲというクラゲは年をとると深海に潜り，約2週間で若返って戻ってくるという．今，世界中の研究者に注目されているとか….未だに詳しい生態は謎らしいですが，不思議ですよね．きっと若返りクラゲ健康法が流行りますよ．紅茶クラゲ，クラゲヨーグルト….（な）

お知らせ

▶**読者の広場**
本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶**投稿歓迎**
本誌に投稿をご希望の方は，連絡先(自宅/勤務先)を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です(送り先はsupportinter@cqpub.co.jpまで)．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶**本誌掲載記事についてのご注意**
本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．
本誌掲載記事をCQ出版(株)の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

▶**コピーサービスのご案内**
本誌バックナンバーの掲載記事については，在庫(原則として24か月分)のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．
●コピー料金(税込み)
1ページにつき100円
●発送手数料(判型に関わらず)
1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
●送付金額の算出方法
総ページ数×100円＋発送手数料
●入金方法
現金書留か郵便小為替による郵送
●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数
●宛て先
〒170-8461　東京都豊島区巣鴨1-14-2
CQ出版株式会社　コピーサービス係
(TEL：03-5395-4211，FAX：03-5395-1642)

▶**お問い合わせ先のご案内**
●在庫，バックナンバー，年間購読送付先変更に関して
販売部：03-5395-2141
●広告に関して
広告部：03-5395-2133
●雑誌本文に関して
編集部：03-5395-2122
記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface
©CQ出版(株)　2003　振替　00100-7-10665
2003年8月号　第29巻　第8号(通巻第134号)
2003年8月1日発行（毎月1日発行）
定価は裏表紙に表示してあります
発行人／蒲生良治
編集人／相原　洋
編集／大野典宏　村上真紀　山口光樹　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森　祐子　唐沢睦子
広告／澤辺　彰　中元正夫　渡部真美
発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1－14－2
電話／編集部(03)5395－2122　URL http://www.cqpub.co.jp/interface/
広告部(03)5395－2133　インターフェース編集部へのメール
販売部(03)5395－2141　supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1－14－2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社
日本ABC協会加盟誌
（新聞雑誌部数公査機構）
ISSN0387-9569
Printed in Japan